

OpenSplice DDS

Version 5.x

C Reference Guide



OpenSplice DDS

C Reference Guide



Part Number: OS-CREFG

Doc Issue 30, 6 January 2011

Copyright Notice

© 2011 PrismTech Limited. All rights reserved.

This document may be reproduced in whole but not in part.

The information contained in this document is subject to change without notice and is made available in good faith without liability on the part of PrismTech Limited or PrismTech Corporation.

All trademarks acknowledged.

A close-up, low-angle photograph of a computer keyboard, focusing on the central and right-hand keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

CONTENTS

Table of Contents

List of Figures	xix
------------------------	------------

Preface

About the C Reference Guide	xxi
Contacts	xxiii

Introduction

About the C Reference Guide	3
Document Structure	3
Operations	4

API Reference

Chapter 1	DCPS API General Description	7
1.1	Thread Safety	8
1.2	Signal Handling	8
1.3	Memory Management	9
1.3.1	IDL Mapping Rules for Sequences	9
1.3.1.1	Standard Defined Type	10
1.3.1.2	User-Defined Type	10
1.3.1.3	Data Distribution Service Defined Type	10
1.3.2	Plain Sequences	10
1.3.3	Sequences Embedded in QosPolicy Objects	11
1.3.4	Sequences Embedded in Status Objects	12
1.3.5	Resources and operations	12
1.3.5.1	Sequences DDS_<sequence-name>	13
1.3.5.2	DDS_sequence_set_release	16
1.4	Listeners Interfaces	22
1.4.1	Struct DDS_<Entity>Listener	24
1.4.2	DDS_DomainParticipantListener	27
1.4.3	DDS_TopicListener	29
1.4.4	DDS_PublisherListener	30
1.4.5	DDS_DataWriterListener	31
1.4.6	DDS_SubscriberListener	32
1.4.7	DDS_DataReaderListener	34
1.5	Inheritance of Abstract Operations	35

Chapter 2	DCPS Modules	37
2.1	Functionality	37
2.2	Infrastructure Module	38
2.3	Domain Module	39
2.4	Topic-Definition Module	40
2.5	Publication Module	42
2.6	Subscription Module	44
Chapter 3	DCPS Classes and Operations	47
3.1	Infrastructure Module	48
3.1.1	Class DDS_Entity (abstract)	48
3.1.1.1	DDS_Entity_enable	49
3.1.1.2	DDS_Entity_get_instance_handle	51
3.1.1.3	DDS_Entity_get_listener (abstract)	52
3.1.1.4	DDS_Entity_get_qos (abstract)	52
3.1.1.5	DDS_Entity_get_status_changes	53
3.1.1.6	DDS_Entity_get_statuscondition	54
3.1.1.7	DDS_Entity_set_listener (abstract)	54
3.1.1.8	DDS_Entity_set_qos (abstract)	55
3.1.2	Class DDS_DomainEntity (abstract)	55
3.1.3	Struct QosPolicy	55
3.1.3.1	DDS_DeadlineQosPolicy	63
3.1.3.2	DDS_DestinationOrderQosPolicy	66
3.1.3.3	DDS_DurabilityQosPolicy	67
3.1.3.4	DDS_DurabilityServiceQosPolicy	71
3.1.3.5	DDS_EntityFactoryQosPolicy	73
3.1.3.6	DDS_GroupDataQosPolicy	74
3.1.3.7	DDS_HistoryQosPolicy	75
3.1.3.8	DDS_LatencyBudgetQosPolicy	77
3.1.3.9	DDS_LifespanQosPolicy	79
3.1.3.10	DDS_LivelinessQosPolicy	80
3.1.3.11	DDS_OwnershipQosPolicy	82
3.1.3.12	DDS_OwnershipStrengthQosPolicy	84
3.1.3.13	DDS_PartitionQosPolicy	85
3.1.3.14	DDS_PresentationQosPolicy	86
3.1.3.15	DDS_ReaderDataLifecycleQosPolicy	92
3.1.3.16	DDS_ReliabilityQosPolicy	93
3.1.3.17	DDS_ResourceLimitsQosPolicy	95
3.1.3.18	DDS_SchedulingQosPolicy	97
3.1.3.19	DDS_TimeBasedFilterQosPolicy	98
3.1.3.20	DDS_TopicDataQosPolicy	98
3.1.3.21	DDS_TransportPriorityQosPolicy	99

3.1.3.22	DDS_UserDataQosPolicy	100
3.1.3.23	DDS_WriterDataLifecycleQosPolicy	100
3.1.4	DDS_Listener interface	101
3.1.5	Struct DDS_Status	102
3.1.5.1	DDS_InconsistentTopicStatus	106
3.1.5.2	DDS_LivelinessChangedStatus	107
3.1.5.3	DDS_LivelinessLostStatus	109
3.1.5.4	DDS_OfferedDeadlineMissedStatus	110
3.1.5.5	DDS_OfferedIncompatibleQosStatus	111
3.1.5.6	DDS_PublicationMatchedStatus	113
3.1.5.7	DDS_RequestedDeadlineMissedStatus	114
3.1.5.8	DDS_RequestedIncompatibleQosStatus	114
3.1.5.9	DDS_SampleLostStatus	116
3.1.5.10	DDS_SampleRejectedStatus	117
3.1.5.11	DDS_SubscriptionMatchedStatus	118
3.1.6	Class DDS_WaitSet	119
3.1.6.1	DDS_WaitSet__alloc	119
3.1.6.2	DDS_WaitSet_attach_condition	120
3.1.6.3	DDS_WaitSet_detach_condition	121
3.1.6.4	DDS_WaitSet_get_conditions	123
3.1.6.5	DDS_WaitSet_wait	124
3.1.7	Class DDS_Condition	126
3.1.7.1	DDS_Condition_get_trigger_value	127
3.1.8	Class DDS_GuardCondition	127
3.1.8.1	DDS_GuardCondition__alloc	128
3.1.8.2	DDS_GuardCondition_get_trigger_value (inherited)	129
3.1.8.3	DDS_GuardCondition_set_trigger_value	129
3.1.9	Class DDS_StatusCondition	130
3.1.9.1	DDS_StatusCondition_get_enabled_statuses	132
3.1.9.2	DDS_StatusCondition_get_entity	133
3.1.9.3	DDS_StatusCondition_get_trigger_value (inherited)	134
3.1.9.4	DDS_StatusCondition_set_enabled_statuses	134
3.2	Domain Module	136
3.2.1	Class DDS_DomainParticipant	136
3.2.1.1	DDS_DomainParticipant_assert_liveliness	141
3.2.1.2	DDS_DomainParticipant_contains_entity	142
3.2.1.3	DDS_DomainParticipant_create_contentfilteredtopic	143
3.2.1.4	DDS_DomainParticipant_create_multitopic	144
3.2.1.5	DDS_DomainParticipant_create_publisher	146
3.2.1.6	DDS_DomainParticipant_create_subscriber	149
3.2.1.7	DDS_DomainParticipant_create_topic	152
3.2.1.8	DDS_DomainParticipant_delete_contained_entities	154

3.2.1.9	DDS_DomainParticipant_delete_contentfilteredtopic	156
3.2.1.10	DDS_DomainParticipant_delete_multitopic	158
3.2.1.11	DDS_DomainParticipant_delete_publisher	159
3.2.1.12	DDS_DomainParticipant_delete_subscriber	160
3.2.1.13	DDS_DomainParticipant_delete_topic	161
3.2.1.14	DDS_DomainParticipant_enable (inherited)	163
3.2.1.15	DDS_DomainParticipant_find_topic	163
3.2.1.16	DDS_DomainParticipant_get_builtin_subscriber	165
3.2.1.17	DDS_DomainParticipant_get_current_time	165
3.2.1.18	DDS_DomainParticipant_get_default_publisher_qos	167
3.2.1.19	DDS_DomainParticipant_get_default_subscriber_qos	168
3.2.1.20	DDS_DomainParticipant_get_default_topic_qos	170
3.2.1.21	DDS_DomainParticipant_get_discovered_participants	171
3.2.1.22	DDS_DomainParticipant_get_discovered_participant_data	171
3.2.1.23	DDS_DomainParticipant_get_discovered_topics	171
3.2.1.24	DDS_DomainParticipant_get_discovered_topic_data	172
3.2.1.25	DDS_DomainParticipant_get_domain_id	172
3.2.1.26	DDS_DomainParticipant_get_listener	173
3.2.1.27	DDS_DomainParticipant_get_qos	173
3.2.1.28	DDS_DomainParticipant_get_status_changes (inherited)	174
3.2.1.29	DDS_DomainParticipant_get_statuscondition (inherited)	175
3.2.1.30	DDS_DomainParticipant_ignore_participant	175
3.2.1.31	DDS_DomainParticipant_ignore_publication	175
3.2.1.32	DDS_DomainParticipant_ignore_subscription	175
3.2.1.33	DDS_DomainParticipant_ignore_topic	176
3.2.1.34	DDS_DomainParticipant_lookup_topicdescription	176
3.2.1.35	DDS_DomainParticipant_set_default_publisher_qos	177
3.2.1.36	DDS_DomainParticipant_set_default_subscriber_qos	178
3.2.1.37	DDS_DomainParticipant_set_default_topic_qos	179
3.2.1.38	DDS_DomainParticipant_set_listener	181
3.2.1.39	DDS_DomainParticipant_set_qos	184
3.2.2	Class DDS_DomainParticipantFactory	185
3.2.2.1	DDS_DomainParticipantFactory_create_participant	187
3.2.2.2	DDS_DomainParticipantFactory_delete_participant	190
3.2.2.3	DDS_DomainParticipantFactory_get_default_participant_qos	191
3.2.2.4	DDS_DomainParticipantFactory_get_instance	193
3.2.2.5	DDS_DomainParticipantFactory_get_qos	193
3.2.2.6	DDS_DomainParticipantFactory_lookup_participant	194
3.2.2.7	DDS_DomainParticipantFactory_set_default_participant_qos	196
3.2.2.8	DDS_DomainParticipantFactory_set_qos	197
3.2.2.9	DDS_DomainParticipantFactory_delete_domain	198
3.2.2.10	DDS_DomainParticipantFactory_lookup_domain	199

3.2.2.11	DDS_DomainParticipantFactory_delete_contained_entities	200
3.2.3	Class DDS_Domain	201
3.2.3.1	DDS_Domain_create_persistent_snapshot	202
3.2.4	DDS_DomainParticipantListener interface	203
3.2.4.1	DDS_DomainParticipantListener__alloc	206
3.2.4.2	DDS_DomainParticipantListener_on_data_available (inherited, abstract)	207
3.2.4.3	DDS_DomainParticipantListener_on_data_on_readers (inherited, abstract)	207
3.2.4.4	DDS_DomainParticipantListener_on_inconsistent_topic (inherited, abstract)	207
3.2.4.5	DDS_DomainParticipantListener_on_liveliness_changed (inherited, abstract)	208
3.2.4.6	DDS_DomainParticipantListener_on_liveliness_lost (inherited, abstract)	208
3.2.4.7	DDS_DomainParticipantListener_on_offered_deadline_missed (inherited, abstract)	208
3.2.4.8	DDS_DomainParticipantListener_on_offered_incompatible_qos (inherited, abstract)	208
3.2.4.9	DDS_DomainParticipantListener_on_publication_matched (inherited, abstract)	209
3.2.4.10	DDS_DomainParticipantListener_on_requested_deadline_missed (inherited, abstract)	209
3.2.4.11	DDS_DomainParticipantListener_on_requested_incompatible_qos (inherited, abstract)	209
3.2.4.12	DDS_DomainParticipantListener_on_sample_lost (inherited, abstract)	210
3.2.4.13	DDS_DomainParticipantListener_on_sample_rejected (inherited, abstract)	210
3.2.4.14	DDS_DomainParticipantListener_on_subscription_matched (inherited, abstract)	210
3.3	Topic-Definition Module	211
3.3.1	Class DDS_TopicDescription (abstract)	212
3.3.1.1	DDS_TopicDescription_get_name	213
3.3.1.2	DDS_TopicDescription_get_participant	213
3.3.1.3	DDS_TopicDescription_get_type_name	214
3.3.2	Class DDS_Topic	215
3.3.2.1	DDS_Topic_enable (inherited)	216
3.3.2.2	DDS_Topic_get_inconsistent_topic_status	216
3.3.2.3	DDS_Topic_get_listener	217
3.3.2.4	DDS_Topic_get_name (inherited)	218
3.3.2.5	DDS_Topic_get_participant (inherited)	218

3.3.2.6	DDS_Topic_get_qos	218
3.3.2.7	DDS_Topic_get_status_changes (inherited)	219
3.3.2.8	DDS_Topic_get_statuscondition (inherited)	220
3.3.2.9	DDS_Topic_get_type_name (inherited)	220
3.3.2.10	DDS_Topic_set_listener	220
3.3.2.11	DDS_Topic_set_qos	222
3.3.3	Class DDS_ContentFilteredTopic	224
3.3.3.1	DDS_ContentFilteredTopic_get_expression_parameters	225
3.3.3.2	DDS_ContentFilteredTopic_get_filter_expression	226
3.3.3.3	DDS_ContentFilteredTopic_get_name (inherited)	227
3.3.3.4	DDS_ContentFilteredTopic_get_participant (inherited)	227
3.3.3.5	DDS_ContentFilteredTopic_get_related_topic	227
3.3.3.6	DDS_ContentFilteredTopic_get_type_name (inherited)	228
3.3.3.7	DDS_ContentFilteredTopic_set_expression_parameters	228
3.3.4	Class DDS_MultiTopic	230
3.3.4.1	DDS_MultiTopic_get_expression_parameters	231
3.3.4.2	DDS_MultiTopic_get_name (inherited)	232
3.3.4.3	DDS_MultiTopic_get_participant (inherited)	232
3.3.4.4	DDS_MultiTopic_get_subscription_expression	233
3.3.4.5	DDS_MultiTopic_get_type_name (inherited)	233
3.3.4.6	DDS_MultiTopic_set_expression_parameters	234
3.3.5	DDS_TopicListener Interface	235
3.3.5.1	DDS_TopicListener__alloc	236
3.3.5.2	DDS_TopicListener_on_inconsistent_topic (abstract)	236
3.3.6	Topic-Definition Type Specific Classes	237
3.3.6.1	Class DDS_TypeSupport (abstract)	238
3.3.6.2	DDS_TypeSupport__alloc (abstract)	239
3.3.6.3	DDS_TypeSupport_get_type_name (abstract)	239
3.3.6.4	DDS_TypeSupport_register_type (abstract)	239
3.3.6.5	Class SPACE_FooTypeSupport	239
3.3.6.6	SPACE_FooTypeSupport__alloc	240
3.3.6.7	SPACE_FooTypeSupport_get_type_name	241
3.3.6.8	SPACE_FooTypeSupport_register_type	242
3.4	Publication Module	244
3.4.1	Class DDS_Publisher	245
3.4.1.1	DDS_Publisher_begin_coherent_changes	247
3.4.1.2	DDS_Publisher_copy_from_topic_qos	249
3.4.1.3	DDS_Publisher_create_datawriter	250
3.4.1.4	DDS_Publisher_delete_contained_entities	254
3.4.1.5	DDS_Publisher_delete_datawriter	255
3.4.1.6	DDS_Publisher_enable (inherited)	256
3.4.1.7	DDS_Publisher_end_coherent_changes	256

3.4.1.8	DDS_Publisher_get_default_datawriter_qos	257
3.4.1.9	DDS_Publisher_get_listener	259
3.4.1.10	DDS_Publisher_get_participant	259
3.4.1.11	DDS_Publisher_get_qos	260
3.4.1.12	DDS_Publisher_get_status_changes (inherited)	261
3.4.1.13	DDS_Publisher_get_statuscondition (inherited)	261
3.4.1.14	DDS_Publisher_lookup_datawriter	261
3.4.1.15	DDS_Publisher_resume_publications	262
3.4.1.16	DDS_Publisher_set_default_datawriter_qos	263
3.4.1.17	DDS_Publisher_set_listener	265
3.4.1.18	DDS_Publisher_set_qos	267
3.4.1.19	DDS_Publisher_suspend_publications	269
3.4.1.20	DDS_Publisher_wait_for_acknowledgments	270
3.4.2	Publication Type Specific Classes	272
3.4.2.1	Class DDS_DataWriter (abstract)	272
3.4.2.2	DDS_DataWriter_assert_liveliness	276
3.4.2.3	DDS_DataWriter_dispose (abstract)	278
3.4.2.4	DDS_DataWriter_dispose_w_timestamp (abstract)	278
3.4.2.5	DDS_DataWriter_enable (inherited)	278
3.4.2.6	DDS_DataWriter_get_key_value (abstract)	279
3.4.2.7	DDS_DataWriter_get_listener	279
3.4.2.8	DDS_DataWriter_get_liveliness_lost_status	279
3.4.2.9	DDS_DataWriter_get_matched_subscription_data	281
3.4.2.10	DDS_DataWriter_get_matched_subscriptions	282
3.4.2.11	DDS_DataWriter_get_offered_deadline_missed_status	284
3.4.2.12	DDS_DataWriter_get_offered_incompatible_qos_status	285
3.4.2.13	DDS_DataWriter_get_publication_matched_status	287
3.4.2.14	DDS_DataWriter_get_publisher	288
3.4.2.15	DDS_DataWriter_get_qos	289
3.4.2.16	DDS_DataWriter_get_status_changes (inherited)	290
3.4.2.17	DDS_DataWriter_get_statuscondition (inherited)	290
3.4.2.18	DDS_DataWriter_get_topic	290
3.4.2.19	DDS_DataWriter_lookup_instance (abstract)	291
3.4.2.20	DDS_DataWriter_register_instance (abstract)	291
3.4.2.21	DDS_DataWriter_register_instance_w_timestamp (abstract)	292
3.4.2.22	DDS_DataWriter_set_listener	292
3.4.2.23	DDS_DataWriter_set_qos	294
3.4.2.24	DDS_DataWriter_unregister_instance (abstract)	296
3.4.2.25	DDS_DataWriter_unregister_instance_w_timestamp (abstract)	296
3.4.2.26	DDS_DataWriter_wait_for_acknowledgments	297
3.4.2.27	DDS_DataWriter_write (abstract)	299
3.4.2.28	DDS_DataWriter_write_w_timestamp (abstract)	299

3.4.2.29	DDS_DataWriter_writedispose (abstract)	299
3.4.2.30	DDS_DataWriter_writedispose_w_timestamp (abstract)	300
3.4.2.31	Class SPACE_FooDataWriter	300
3.4.2.32	SPACE_FooDataWriter_assert_liveliness (inherited)	304
3.4.2.33	SPACE_FooDataWriter_dispose	304
3.4.2.34	SPACE_FooDataWriter_dispose_w_timestamp	308
3.4.2.35	SPACE_FooDataWriter_enable (inherited)	310
3.4.2.36	SPACE_FooDataWriter_get_key_value	310
3.4.2.37	SPACE_FooDataWriter_get_listener (inherited)	311
3.4.2.38	SPACE_FooDataWriter_get_liveliness_lost_status (inherited)	312
3.4.2.39	SPACE_FooDataWriter_get_matched_subscription_data (inherited)	312
3.4.2.40	SPACE_FooDataWriter_get_matched_subscriptions (inherited)	312
3.4.2.41	SPACE_FooDataWriter_get_offered_deadline_missed_status (inherited)	312
3.4.2.42	SPACE_FooDataWriter_get_offered_incompatible_qos_status (inherited)	313
3.4.2.43	SPACE_FooDataWriter_get_publication_matched_status (inherited)	313
3.4.2.44	SPACE_FooDataWriter_get_publisher (inherited)	313
3.4.2.45	SPACE_FooDataWriter_get_qos (inherited)	313
3.4.2.46	SPACE_FooDataWriter_get_status_changes (inherited)	314
3.4.2.47	SPACE_FooDataWriter_get_statuscondition (inherited)	314
3.4.2.48	SPACE_FooDataWriter_get_topic (inherited)	314
3.4.2.49	SPACE_FooDataWriter_lookup_instance	314
3.4.2.50	SPACE_FooDataWriter_register_instance	316
3.4.2.51	SPACE_FooDataWriter_register_instance_w_timestamp	318
3.4.2.52	SPACE_FooDataWriter_set_listener (inherited)	319
3.4.2.53	SPACE_FooDataWriter_set_qos (inherited)	320
3.4.2.54	SPACE_FooDataWriter_unregister_instance	320
3.4.2.55	SPACE_FooDataWriter_unregister_instance_w_timestamp	324
3.4.2.56	SPACE_FooDataWriter_wait_for_acknowledgments (inherited)	325
3.4.2.57	SPACE_FooDataWriter_write	326
3.4.2.58	SPACE_FooDataWriter_write_w_timestamp	329
3.4.2.59	SPACE_FooDataWriter_writedispose	330
3.4.2.60	SPACE_FooDataWriter_writedispose_w_timestamp	335
3.4.3	DDS_PublisherListener interface	336
3.4.3.1	DDS_PublisherListener__alloc	338
3.4.3.2	DDS_PublisherListener_on_liveliness_lost (inherited, abstract)	338
3.4.3.3	DDS_PublisherListener_on_offered_deadline_missed (inherited, abstract)	339
3.4.3.4	DDS_PublisherListener_on_offered_incompatible_qos	

xiii
C Reference Guide

3.5.2.12	DDS_DataReader_get_qos	388
3.5.2.13	DDS_DataReader_get_requested_deadline_missed_status	389
3.5.2.14	DDS_DataReader_get_requested_incompatible_qos_status	390
3.5.2.15	DDS_DataReader_get_sample_lost_status	391
3.5.2.16	DDS_DataReader_get_sample_rejected_status	393
3.5.2.17	DDS_DataReader_get_status_changes (inherited).	394
3.5.2.18	DDS_DataReader_get_statuscondition (inherited)	394
3.5.2.19	DDS_DataReader_get_subscriber	394
3.5.2.20	DDS_DataReader_get_subscription_matched_status	395
3.5.2.21	DDS_DataReader_get_topicdescription.	396
3.5.2.22	DDS_DataReader_lookup_instance (abstract).	397
3.5.2.23	DDS_DataReader_read (abstract)	397
3.5.2.24	DDS_DataReader_read_instance (abstract).	398
3.5.2.25	DDS_DataReader_read_next_instance (abstract)	398
3.5.2.26	DDS_DataReader_read_next_instance_w_condition (abstract)	399
3.5.2.27	DDS_DataReader_read_next_sample (abstract)	399
3.5.2.28	DDS_DataReader_read_w_condition (abstract)	399
3.5.2.29	DDS_DataReader_return_loan (abstract).	400
3.5.2.30	DDS_DataReader_set_listener.	400
3.5.2.31	DDS_DataReader_set_qos	403
3.5.2.32	DDS_DataReader_take (abstract).	405
3.5.2.33	DDS_DataReader_take_instance (abstract).	405
3.5.2.34	DDS_DataReader_take_next_instance (abstract).	405
3.5.2.35	DDS_DataReader_take_next_instance_w_condition (abstract).	406
3.5.2.36	DDS_DataReader_take_next_sample (abstract)	406
3.5.2.37	DDS_DataReader_take_w_condition (abstract).	407
3.5.2.38	DDS_DataReader_wait_for_historical_data	407
3.5.2.39	Class SPACE_FooDataReader.	409
3.5.2.40	SPACE_FooDataReader_create_querycondition (inherited).	415
3.5.2.41	SPACE_FooDataReader_create_readcondition (inherited).	415
3.5.2.42	SPACE_FooDataReader_delete_contained_entities (inherited)	415
3.5.2.43	SPACE_FooDataReader_delete_readcondition (inherited).	415
3.5.2.44	SPACE_FooDataReader_enable (inherited)	416
3.5.2.45	SPACE_FooDataReader_get_key_value	416
3.5.2.46	SPACE_FooDataReader_get_listener (inherited)	417
3.5.2.47	SPACE_FooDataReader_get_liveliness_changed_status (inherited)	417
3.5.2.48	SPACE_FooDataReader_get_matched_publication_data (inherited)	418
3.5.2.49	SPACE_FooDataReader_get_matched_publications (inherited).	418
3.5.2.50	SPACE_FooDataReader_get_qos (inherited)	418
3.5.2.51	SPACE_FooDataReader_get_requested_deadline_missed_status (inherited)	418
3.5.2.52	SPACE_FooDataReader_get_requested_incompatible_qos_status	

(inherited)	419
3.5.2.53 SPACE_FooDataReader_get_sample_lost_status (inherited)	419
3.5.2.54 SPACE_FooDataReader_get_sample_rejected_status (inherited)	419
3.5.2.55 SPACE_FooDataReader_get_status_changes (inherited)	419
3.5.2.56 SPACE_FooDataReader_get_statuscondition (inherited)	420
3.5.2.57 SPACE_FooDataReader_get_subscriber (inherited)	420
3.5.2.58 SPACE_FooDataReader_get_subscription_matched_status (inherited)	420
3.5.2.59 SPACE_FooDataReader_get_topicdescription (inherited)	420
3.5.2.60 SPACE_FooDataReader_lookup_instance.	421
3.5.2.61 SPACE_FooDataReader_read	421
3.5.2.62 SPACE_FooDataReader_read_instance.	426
3.5.2.63 SPACE_FooDataReader_read_next_instance	428
3.5.2.64 SPACE_FooDataReader_read_next_instance_w_condition	431
3.5.2.65 SPACE_FooDataReader_read_next_sample	433
3.5.2.66 SPACE_FooDataReader_read_w_condition	433
3.5.2.67 SPACE_FooDataReader_return_loan	435
3.5.2.68 SPACE_FooDataReader_set_listener (inherited).	438
3.5.2.69 SPACE_FooDataReader_set_qos (inherited).	438
3.5.2.70 SPACE_FooDataReader_take	438
3.5.2.71 SPACE_FooDataReader_take_instance.	440
3.5.2.72 SPACE_FooDataReader_take_next_instance	442
3.5.2.73 SPACE_FooDataReader_take_next_instance_w_condition	444
3.5.2.74 SPACE_FooDataReader_take_next_sample	446
3.5.2.75 SPACE_FooDataReader_take_w_condition	446
3.5.2.76 SPACE_FooDataReader_wait_for_historical_data (inherited)	448
3.5.3 Class DDS_DataSample	448
3.5.4 Struct DDS_SampleInfo	448
3.5.4.1 DDS_SampleInfo	449
3.5.5 DDS_SubscriberListener Interface	453
3.5.5.1 DDS_SubscriberListener__alloc	454
3.5.5.2 DDS_SubscriberListener_on_data_available (inherited, abstract).	455
3.5.5.3 DDS_SubscriberListener_on_data_on_readers (abstract)	455
3.5.5.4 DDS_SubscriberListener_on_liveliness_changed (inherited, abstract)	457
3.5.5.5 DDS_SubscriberListener_on_requested_deadline_missed (inherited, abstract)	457
3.5.5.6 DDS_SubscriberListener_on_requested_incompatible_qos (inherited, abstract)	457
3.5.5.7 DDS_SubscriberListener_on_sample_lost (inherited, abstract).	457
3.5.5.8 DDS_SubscriberListener_on_sample_rejected (inherited, abstract)	458
3.5.5.9 DDS_SubscriberListener_on_subscription_matched	

(inherited, abstract)	458
3.5.6 DDS_DataReaderListener interface	458
3.5.6.1 DDS_DataReaderListener__alloc	460
3.5.6.2 DDS_DataReaderListener_on_data_available (abstract)	461
3.5.6.3 DDS_DataReaderListener_on_liveliness_changed (abstract)	462
3.5.6.4 DDS_DataReaderListener_on_requested_deadline_missed (abstract)	463
3.5.6.5 DDS_DataReaderListener_on_requested_incompatible_qos (abstract)	464
3.5.6.6 DDS_DataReaderListener_on_sample_lost (abstract)	465
3.5.6.7 DDS_DataReaderListener_on_sample_rejected (abstract)	465
3.5.6.8 DDS_DataReaderListener_on_subscription_matched (abstract)	466
3.5.7 Class DDS_ReadCondition	467
3.5.7.1 DDS_ReadCondition_get_datareader	468
3.5.7.2 DDS_ReadCondition_get_instance_state_mask	469
3.5.7.3 DDS_ReadCondition_get_sample_state_mask	470
3.5.7.4 DDS_ReadCondition_get_trigger_value (inherited)	470
3.5.7.5 DDS_ReadCondition_get_view_state_mask	471
3.5.8 Class DDS_QueryCondition	471
3.5.8.1 DDS_QueryCondition_get_datareader (inherited)	473
3.5.8.2 DDS_QueryCondition_get_instance_state_mask (inherited)	473
3.5.8.3 DDS_QueryCondition_get_query_parameters	473
3.5.8.4 DDS_QueryCondition_get_query_expression	474
3.5.8.5 DDS_QueryCondition_get_sample_state_mask (inherited)	475
3.5.8.6 DDS_QueryCondition_get_trigger_value (inherited)	475
3.5.8.7 DDS_QueryCondition_get_view_state_mask (inherited)	476
3.5.8.8 DDS_QueryCondition_set_query_parameters	476
Appendix A Quality Of Service	481
Affected Entities	481
Basic Usage	481
DDS_DataReaderQos	483
DDS_DataWriterQos	486
DDS_DomainParticipantFactoryQos	488
DDS_DomainParticipantQos	489
DDS_PublisherQos	491
DDS_SubscriberQos	492
DDS_TopicQos	493
Appendix B API Constants and Types	497
Appendix C Platform Specific IDL Interface	501
dds_dcps.idl	501

Appendix D	SampleStates, ViewStates and InstanceStates	529
	SampleInfo Class	529
	sample_state	529
	instance_state	531
	view_state	532
	State Masks	534
	Operations Concerning States	535
Appendix E	Class Inheritance	539
Appendix F	Listeners, Conditions and Waitsets	541
	Communication Status Event	543
	Listeners	546
	Conditions and Waitsets	547
	DDS_StatusCondition Trigger State	551
	DDS_ReadCondition and DDS_QueryCondition Trigger State	551
	DDS_GuardCondition Trigger State	552
Appendix G	DDS_Topic Definitions	553
	DDS_Topic Definition Example	553
	Complex Topics	554
	IDL Pre-processor	554
Appendix H	DCPS Queries and Filters	559
	SQL Grammar in BNF	559
	SQL Token Expression	560
	SQL Examples	561
	Bibliography	565
	Glossary	569
	Index	573

List of Figures

Figure 1 C Reference Guide Document Structure	3
Figure 2 DCPS Module Composition	37
Figure 3 DCPS Infrastructure Module's Class Model	38
Figure 4 DCPS Domain Module's Class Model	40
Figure 5 DCPS Topic-Definition Module's Class Model	41
Figure 6 Data Type "Foo" Typed Classes Pre-processor Generation	42
Figure 7 DCPS Publication Module's Class Model	43
Figure 8 DCPS Subscription Module's Class Model	44
Figure 9 DCPS Infrastructure Module's Class Model	48
Figure 10 QosPolicy Settings	56
Figure 11 DCPS Listeners	102
Figure 12 DCPS DDS_Status Values	105
Figure 13 DCPS DDS_WaitSets	119
Figure 14 DCPS DDS_Conditions	126
Figure 15 DCPS Domain Module's Class Model	136
Figure 16 DCPS Topic-Definition Module Class Model	211
Figure 17 Pre-processor Generation of the Typed Classes for Data Type "Foo"	212
Figure 18 The DCPS Publication Module's Class Model	244
Figure 19 The DCPS Subscription Module's Class Model	347
Figure 20: State Chart of the sample_state for a Single Sample	530
Figure 21: State Chart of the instance_state for a Single Instance	532
Figure 22: State Chart of the view_state for a Single Instance	533
Figure 23 DCPS Inheritance	539
Figure 24: Plain Communication Status State Chart	544
Figure 25: Read Communication Status DDS_DataReader Statecraft	545
Figure 26: DDS_Subscriber Statecraft for a Read Communication Status .	546
Figure 27: DCPS Listeners	547
Figure 28: DCPS DDS_WaitSets	548
Figure 29 DCPS DDS_Conditions	550
Figure 30: Blocking Behaviour of a Waitset State Chart	551

Preface

About the C Reference Guide

The *C Reference Guide* provides a detailed explanation of the OpenSplice DDS (*Subscription Paradigm for the Logical Interconnection of Concurrent Engines*) Application Programming Interfaces for the C language.

This reference guide is based on the OMG's *Data Distribution Service Specification* and *C Language Mapping Specification*.

The C Reference Guide focuses on the Data Centric Publish Subscribe (DCPS) layer and does not cover the DLRL layer. The purpose of the DCPS is the distribution of data (publish/subscribe). The structure of the DCPS is divided into five modules. Each module consists of several classes, which in turn generally contain several operations.

Intended Audience

The *C Reference Guide* is intended to be used by C programmers who are using OpenSplice DDS to develop applications.

Organisation

The C Reference Guide is organised into the following topics.

The *Introduction* describes the details of the document structure.

Chapter 1, *DCPS API General Description*, is a general description of the DCPS API and its error codes.

Chapter 2, *DCPS Modules*, provides the detailed description of the DCPS modules.

Chapter 3, *DCPS Classes and Operations*, provides the detailed description of the DCPS classes, structs and operations.

The following appendices are included, as well as a *Bibliography* containing references material and *Glossary*:

Appendix A, *Quality Of Service*

Appendix B, *API Constants and Types*

Appendix C, *Platform Specific IDL Interface*

Appendix D, *SampleStates, ViewStates and InstanceStates*

Appendix E, *Class Inheritance*

Appendix F, *Listeners, Conditions and Waitsets*

Appendix G, *DDS_Topic Definitions*

Appendix H, *DCPS Queries and Filters*

Conventions

The conventions listed below are used to guide and assist the reader in understanding the C Reference Guide.



Item of special significance or where caution needs to be taken.



Item contains helpful hint or special information.



Information applies to Windows (*e.g.* XP, 2003, Windows 7) only.



Information applies to Unix based systems (*e.g.* Solaris) only.



C language specific



C++ language specific



Java language specific

Hypertext links are shown as *[blue italic underlined](#)*.

On-Line (PDF) versions of this document: Items shown as cross references, *e.g.* *Contacts* on page xxiii, are hypertext links: click on the reference to go to the item.

```
% Commands or input which the user enters on the
   command line of their computer terminal
```

Courier fonts indicate programming code and file names.

Extended code fragments are shown in shaded boxes:

```
NameComponent newName[] = new NameComponent[1];

// set id field to "example" and kind field to an empty string
newName[0] = new NameComponent ("example", "");
```

Italics and ***Italic Bold*** are used to indicate new terms, or emphasise an item.

Arial Bold is used to indicate user related actions, *e.g.* **File > Save** from a menu.

Step 1: One of several steps required to complete a task.

Contacts

PrismTech can be reached at the following contact points for information and technical support.

USA Corporate Headquarters

PrismTech Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA

Tel: +1 781 569 5819

European Head Office

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK

Tel: +44 (0)191 497 9900

Fax: +44 (0)191 497 9901

Web:

<http://www.prismtech.com>

Technical questions:

crc@prismtech.com (Customer Response Center)

Sales enquiries:

sales@prismtech.com

A close-up, low-angle shot of a computer keyboard, focusing on the keys. The image is overlaid with a white grid pattern, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

INTRODUCTION

About the C Reference Guide

Document Structure

The C Reference Guide document structure is based on the structure of the DCPS Platform Independent Model (DCPS PIM) of the Data Distribution Service Specification. The detailed description is subdivided into the PIM Modules, which are then subdivided into classes.

Some of the classes are implemented as structs in the DCPS Platform Specific Model (DCPS PSM) of the Data Distribution Service Specification, as indicated in the Interface Description Language (IDL) chapter of the PSM (see Appendix C, *Platform Specific IDL Interface*). These structs are described in the respective chapters.

- In the classes as described in the PIM, which are implemented as a class in the PSM, the operations are described in detail.
- In the classes as described in the PIM, which are implemented as a struct in the PSM, the struct contents are described in detail.
- The order of the modules and classes is conform the PIM part.
- The order of the operations or struct contents is alphabetical.
- Each description of a class or struct starts with the API description header file.

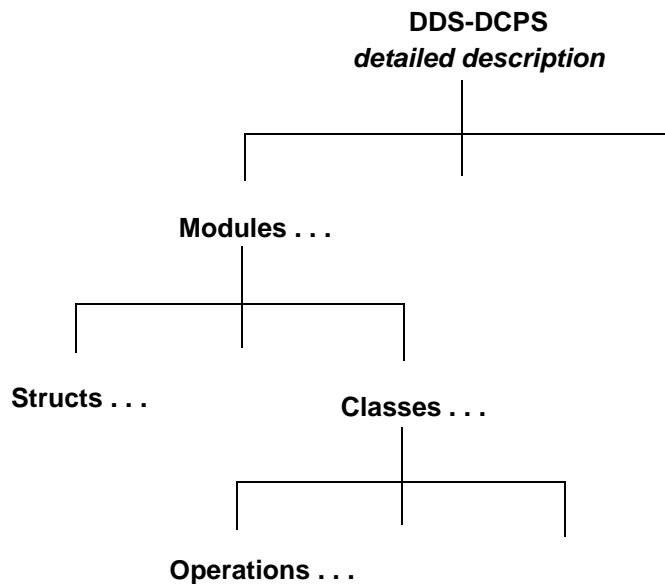


Figure 1 C Reference Guide Document Structure

Operations

Several types of operations are described in this manual. The different types of operations are: basic, inherited, abstract and abstract interface. All operations of any type can be found in their respective class. The details of their description depends on the type of operation.

Basic operations are described in detail in the class they are implemented in.

- Inherited operations only refer to the operation in the class they are inherited from. The detailed description is not repeated.
- Abstract operations only refer to the type specific implementations in their respective derived class. The detailed description is not repeated.
- Abstract operations which are implemented as an interface (`Listeners`), are described in detail in their class. These operations must be implemented in the application.

In the API description header file, the inherited and abstract operations are commented out since they are not implemented in this class.

Inheritance in the C API is implemented by prefixing the name of the operation with `DDS_` and the name of the class they are in. For example, the operation `get_name` in the class `Topic` is named `DDS_Topic_get_name`. Since this operation is actually inherited from the class `TopicDescription` the operation refers to the `TopicDescription` class for more information. However, in the `TopicDescription` class this operation is named `DDS_TopicDescripton_get_name`.

A close-up, low-angle shot of a computer keyboard, focusing on the central and right-hand keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

API REFERENCE

CHAPTER

1 *DCPS API General Description*

The structure of the DCPS is divided into modules, which are described in detail in the next chapter. Each module consists of several classes, which in turn may contain several operations.

Some of these operations have an operation return code of type `DDS_ReturnCode_t`, which is defined in the next table:

Table 1 Return Codes

DDS_ReturnCode_t	Return Code Description
<code>DDS_RETCODE_OK</code>	Successful return
<code>DDS_RETCODE_ERROR</code>	Generic, unspecified error
<code>DDS_RETCODE_BAD_PARAMETER</code>	Illegal parameter value
<code>DDS_RETCODE_UNSUPPORTED</code>	Unsupported operation or <code>DDS_QosPolicy</code> setting. Can only be returned by operations that are optional or operations that uses an optional <code>DDS_<DDS_Entity>QoS</code> as a parameter
<code>DDS_RETCODE_ALREADY_DELETED</code>	The object target of this operation has already been deleted
<code>DDS_RETCODE_OUT_OF_RESOURCES</code>	Service ran out of the resources needed to complete the operation
<code>DDS_RETCODE_NOT_ENABLED</code>	Operation invoked on an <code>DDS_Entity</code> that is not yet enabled
<code>DDS_RETCODE_IMMUTABLE_POLICY</code>	Application attempted to modify an immutable <code>DDS_QosPolicy</code>
<code>DDS_RETCODE_INCONSISTENT_POLICY</code>	Application specified a set of policies that are not consistent with each other
<code>DDS_RETCODE_PRECONDITION_NOT_MET</code>	A pre-condition for the operation was not met

Table 1 Return Codes

DDS_ReturnCode_t	Return Code Description
DDS_RETCODE_TIMEOUT	The operation timed out
DDS_RETCODE_ILLEGAL_OPERATION	An operation was invoked on an inappropriate object or at an inappropriate time (as determined by QosPolicies that control the behaviour of the object in question). There is no precondition that could be changed to make the operation succeed.
DDS_RETCODE_NO_DATA	Indicates a situation where the operation did not return any data

The name scope (name space) of these return codes is DDS. The operation return codes `DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_UNSUPPORTED` and `DDS_RETCODE_ALREADY_DELETED` are default for operations that return an operation return code and are therefore not explicitly mentioned in the DDS specification. However, in this manual they are mentioned along with each operation.

Some operations are not implemented. These operations are mentioned including their synopsis, but not described in this manual and return `DDS_RETCODE_UNSUPPORTED` when called from the application. All constants and types are given in Appendix B, *API Constants and Types*.

1.1 Thread Safety

All operations are thread safe apart from the `DDS_DomainParticipantFactory_get_instance` operation. It is the applications responsibility to call `DDS_DomainParticipantFactory_get_instance` only from one application thread. This restriction only applies to the first call of `DDS_DomainParticipantFactory_get_instance`.

1.2 Signal Handling

UNIX

The Data Distribution Service sets signal handlers in order to assure that resources are released when signals that terminate the application process are cached. These signal handlers only call the exit function in order to force exit handlers to be activated.

If the application needs to set signal handlers for its own use, two situations can occur. In the first case the application sets a signal handler for a specific signal while the Data Distribution Service has not set a handler yet. The Data Distribution Service will not set its own handler in this case, but expects the application signal handler to call the exit function when the signal is meant to terminate the process. In

the second case the Data Distribution Service has already set a signal handler for a specific signal and the application program redefines the signal handling by setting its own handler. In that case the application should either chain the Data Distribution Service signal handler (to be executed as last) or to call the exit function itself when the cached signal is meant to terminate the application process.

The Data Distribution Service service will conditionally set the signal handlers when creating the `DomainParticipantFactory`, which is the first call to `DDS_DomainParticipantFactory_get_instance` for C.

The Data Distribution Service only sets signal handlers for signals that have the default behaviour of terminating the process without dumping a core.

1.3 Memory Management

When objects are being created, they occupy memory space. To avoid memory leaks when they are not used any more, these objects have to be deleted in order to release the memory space. However, when using pointers, it is difficult to keep track of which object has been released and which has not. When objects are not being released, the memory leak finally uses up all the resources and the application fails. On the other hand, when an object is being released twice because there were two pointers to the same object, the application fails. This implementation is based on the *OMG C Language Mapping Specification*. Accordingly, the CORBA rules listed below apply.

1.3.1 IDL Mapping Rules for Sequences

The names of the operations and types are given by the IDL mapping rules. For sequences several rules apply. The basic IDL definition of a sequence is defined by:

```
module name-space {
    typedef sequence<<sequence-element-type>>
    <sequence-name>;
}
```

In the C language, this results in the following type definition of the sequence:

```
typedef
    DDS_sequence_<name-space-prefix><sequence-element-type>
    <name-space>_<sequence-name>
```

In this type definition, the `<sequence-element-type>` is the type of the objects in the sequence. This `<sequence-element-type>` may be a standard type or a Data Distribution Service defined type. The `<name-space-prefix>` represents the name space in which the `<sequence-element-type>` is defined. The standard types have an empty prefix. In the Data Distribution Service all the

typedefs are set within the module DDS block, therefore defined types have the prefix DDS_. Finally, the <sequence-name> is name of the sequence and is always prefixed by DDS_.

1.3.1.1 Standard Defined Type

The standard defined types are the types as defined in the Data Distribution Service specification. For example, for the standard defined <sequence-element-type> of type string with a <sequence-name> of StringSeq, the following IDL definition is given:

```
typedef sequence<string> StringSeq
```

In C, this results in the following type definition of the sequence:

```
typedef DDS_sequence_string DDS_StringSeq
```

1.3.1.2 User-Defined Type

The user-defined types are the types as defined in the application. For example, for the user-defined <sequence-element-type> of type Foo with a <sequence-name> of name FooSeq in the module SPACE, the following IDL definition is given:

```
module SPACE {
    typedef sequence<Foo> FooSeq;
}
```

In C, this results in the following type definition of the sequence:

```
typedef DDS_sequence_SPACE_Foo SPACE_FooSeq
```

1.3.1.3 Data Distribution Service Defined Type

For example, for the Data Distribution Service defined <sequence-element-type> of type SampleInfo with a <sequence-name> of name SampleInfoSeq, the following IDL definition is given:

```
typedef sequence<SampleInfo> SampleInfoSeq
```

In C, this results in the following type definition of the sequence:

```
typedef DDS_sequence_DDS_SampleInfo DDS_SampleInfoSeq
```

1.3.2 Plain Sequences

The following table shows the sequences for which the resources have to be managed. In other words, for these sequences DDS_<sequence-name>__alloc and DDS_<sequence-name>_allocbuf operations are available. For sequences, which are only used as an out parameter, the application does not need to use these allocation operations, since the Data Distribution Service allocates them. In this case, the application may use these operations for its own sequences. Furthermore to free the resources allocated with DDS_<sequence-name>__alloc and

DDS_<sequence-name>_allocbuf the application must use the DDS_free operation. It does not make any difference whether the application or the Data Distribution Service does the allocation. When the application does not use the DDS_free operation, the application will fail. The DDS_free operation operates recursively, in other words all embedded structures are released.

Sequences and buffers can also be allocated on stack. However in case the application allocates a sequence or buffer on stack, the DDS_free operation may not be used on this object, otherwise the application will fail.

Table 2 Sequences

Sequence Name	Parameter Type			
	In	Out	Inout	Return
DDS_ConditionSeq		X		
DDS_StringSeq	X			X
DDS_DataReaderSeq		X		
DDS_InstanceHandleSeq			X	
DDS_QosPolicyCountSeq	Used in status struct only.			
DDS_SampleInfoSeq			X	
DDS_sequence_octet	Used in QosPolicy struct only.			

1.3.3 Sequences Embedded in QosPolicy Objects

The following table shows the QosPolicy objects for which the resources have to be managed because they contain sequences. In other words, for these QosPolicy objects DDS_<QosPolicy>__alloc operations are available. The buffers used in these QosPolicy objects must be allocated using the DDS_<sequence-name>_allocbuf operations. The DDS_free operation takes care of the embedded sequences and the buffers in a QosPolicy.

Table 3 QosPolicy Objects

QosPolicy Name	Parameter Type				Contains Sequence
	In	Out	Inout	Return	
DDS_DomainParticipantQos	X		X		DDS_sequence_octet
DDS_TopicQos	X		X		DDS_sequence_octet
DDS_PublisherQos	X		X		DDS_sequence_octet
					DDS_StringSeq
DDS_DataWriterQos	X		X		DDS_sequence_octet

Table 3 QosPolicy Objects

QosPolicy Name	Parameter Type				Contains Sequence
	In	Out	Inout	Return	
DDS_SubscriberQos	X		X		DDS_sequence_octet
					DDS_StringSeq
DDS_DataReaderQos	X		X		DDS_sequence_octet

1.3.4 Sequences Embedded in Status Objects

The following table shows the Status objects for which the resources have to be managed because they contain sequences. In other words, for these Status objects DDS_<Status>__alloc operations are available. The buffers used in these Status objects must be allocated using the DDS_<sequence-name>_allocbuf operations. The DDS_free operation takes care of the embedded sequences and the buffers in a Status.

Table 4 Status Objects

Status Name	Parameter Type				Contains Sequence
	In	Out	Inout	Return	
DDS_OfferedIncompatibleQosStatus	X		X		DDS_QosPolicyCountSeq
DDS_RequestedIncompatibleQosStatus	X		X		DDS_QosPolicyCountSeq

1.3.5 Resources and operations

The interface description of the memory management operations is as follows:

```

/* interface Memory management */
typedef struct {
    DDS_unsigned_long _maximum;
    DDS_unsigned_long _length;
    DDS_<sequence-element-type> *_buffer;
    DDS_boolean _release;
} DDS_sequence_<name-space-prefix><sequence-element-type>;
typedef
    DDS_sequence_<name-space-prefix><sequence-element-type>
    DDS_<sequence-name>

/* implemented API operations */
void
    DDS_sequence_set_release
        (void *sequence,
         DDS_boolean release);
DDS_boolean

```

```

        DDS_sequence_get_release
            (void *sequence);
DDS_<sequence-name> *
    DDS_<sequence-name>__alloc
        (void);
DDS_<sequence-element-type> *
    DDS_<sequence-name>__allocbuf
        (DDS_unsigned_long len);
DDS_<QosPolicy>
    DDS_<QosPolicy>__alloc
        (void);
DDS_<Status>
    DDS_<Status>__alloc
        (void);
void
    DDS_free
        (void *);

```

The following paragraphs describe the usage of all memory management operations.

1.3.5.1 Sequences DDS_<sequence-name>

Synopsis

```

#include <dds_dcps.h>
typedef struct {
    DDS_unsigned_long _maximum;
    DDS_unsigned_long _length;
    DDS_<sequence-element-type> *_buffer;
    DDS_boolean _release;
} DDS_sequence_<name-space-prefix>
<sequence-element-type>;
typedef DDS_sequence_<name-space-prefix>
    <sequence-element-type> DDS_<sequence-name>

```

Description

The typedef DDS_<sequence-name> represents the sequence which contains the objects of <sequence-element-type>.

Attributes

DDS_unsigned_long _maximum - the maximum number of elements that can be contained in the sequence.

DDS_unsigned_long _length - the actual number of elements in the sequence.

DDS_<sequence-element-type> *_buffer - a pointer to the sequence buffer.

DDS_boolean _release - indicates whether this sequence owns the memory of *_buffer*.

Detailed Description

The typedef `DDS_<sequence-name>` represents the sequence struct that holds the sequence attributes associated with the sequence buffer, which contains the objects of `<sequence-element-type>`. This sequence is allocated by calling `DDS_<sequence-name>__alloc`. The sequence buffer must be allocated separately by calling `DDS_<sequence-name>_allocbuf`. In other words when using a sequence, the memory space must be allocated for both the sequence struct and the sequence buffer. Whether, the application must allocate the resources or the Data Distribution Service allocates the resources, depends on the type of usage.

In or Inout Parameter

In case the sequence is passed as an in or inout parameter, both the sequence and the buffer must be allocated by the application. The application must set the attributes of the sequence according to the size and ownership of the buffer. Furthermore, for an inout parameter the application can control whether the Data Distribution Service must replace the elements in the sequence, the application can allow this by setting the *_release* attribute.

- When set to `TRUE` the Data Distribution Service is allowed to free any pointer types. The Data Distribution Service sets the *_length* attribute to the number of returned elements. The number of elements never exceeds the number set by the application in the *_maximum* attribute.
- When set to `FALSE` the Data Distribution Service is not allowed to free the pointer types. In this case, the Data Distribution Service allocates exactly the amount of elements and set the *_length* and the *_maximum* attributes of the sequence to that amount.
- In either case, the sequence and the buffer must be released by the application by calling `DDS_free` on the sequence. In this case also the buffer is released, since the `DDS_free` operation is recursive.

Out or Return Parameter

In case the sequence is used as an out parameter or a sequence is returned by a function, both the sequence and the buffer are allocated by the Data Distribution Service. The attributes of the sequence are set by the Data Distribution Service according to the size and ownership of the buffer. The sequence and the buffer must be released by the application by calling `DDS_free` on the sequence. In this case also the buffer is released, since the `DDS_free` operation is recursive.

In case the Data Distribution Service has no data to return, it returns an empty sequence with the `_length` and the `_maximum` attributes of the sequence set to zero, the `_buffer` attribute set to `DDS_OBJECT_NIL` and the `_release` attribute set to `FALSE`.

Allocation on the Stack

In case the sequence is allocated by the application. The application may also allocate the sequence on stack for performance reason instead of calling `DDS_<sequence-name>__alloc`. When the buffer is allocated on the stack the application must also set the `_release` attribute to `FALSE` as described below. In case the buffer is allocated using `DDS_<sequence-name>_allocbuf` then the application must release the buffer separately by calling `DDS_free` on `_buffer` of the sequence.

Attributes

The attributes of the `DDS_<sequence-name>` struct must be set after allocation. In case of an out parameter or the sequence is returned by a function, the attributes are set by the Data Distribution Service. In case of an in parameter or inout parameter, the attributes must be set by the application.

The `_length` attribute of the sequence must be set to the current length of the sequence. In other words equal to the number of valid sequence elements.

The `_maximum` attribute of the sequence must be set to the size of the allocated sequence buffer. In other words equal to the `len` parameter used in the call to `DDS_<sequence-element-type>_allocbuf`.

The `_buffer` attribute of the sequence must be set to the pointer to the allocated sequence buffer. In other words equal to the returned pointer from the call to `DDS_<sequence-element-type>_allocbuf`. Or in case of allocation on stack, the pointer to the variable.

The `_release` flag of the sequence may not be set directly. The `_release` flag of the sequence must be set by using `DDS_sequence_set_release` and may only be read by using `DDS_sequence_get_release`. `DDS_sequence_set_release` may only be used by the creator of the sequence. If it is not called for a given sequence instance, then the default value of the `_release` flag for that instance is `FALSE`.

If the `_release` flag of the sequence is set to `TRUE`, the sequence effectively “owns” the resource pointed to by `_buffer`; if the flag is set to `FALSE`, the application is responsible for the resource. If, for example, a sequence is returned from an operation with its release flag set to `FALSE`, calling `DDS_free` on the returned sequence pointer does not deallocate the memory pointed to by `_buffer`.

Before calling `DDS_free` on the `_buffer` member of a sequence directly, the application should check the `_release` flag using `DDS_sequence_get_release`. If it returns `FALSE`, the application should not invoke `DDS_free` on the `_buffer` member; doing so produces undefined behaviour.

1.3.5.2 DDS_sequence_set_release

Synopsis

```
#include <dds_dcps.h>
void
    DDS_sequence_set_release
        (void *sequence, DDS_boolean release);
```

Description

This operation sets the state of the `_release` flag of the sequence.

Parameters

in void *sequence - a pointer to the `DDS_<sequence-name>`.

in DDS_boolean release - the new state of the `_release` flag of the sequence.

Return Value

<none>

Detailed Description

This operation sets the state of the `_release` flag of the sequence. If the flag is set to `TRUE`, the sequence effectively “owns” the resource pointed to by `_buffer`; if the flag is set to `FALSE`, the application is responsible for the resource. If, for example, a sequence is returned from an operation with its release flag set to `FALSE`, calling `DDS_free` on the returned sequence pointer does not deallocate the memory pointed to by `_buffer`. Passing a `DDS_OBJECT_NIL` pointer or a pointer to something other than a sequence type to `DDS_sequence_set_release` produces undefined behaviour.

`DDS_sequence_set_release` should only be used by the creator of the sequence. If it is not called for a given sequence instance, then the default value of the `_release` flag for that instance is `FALSE`. The `_release` flag of the sequence may not be set directly. It may only be changed by this operation.

1.3.5.2.1 DDS_sequence_get_release

Synopsis

```
#include <dds_dcps.h>
DDS_boolean
    DDS_sequence_get_release
        (void *sequence);
```

Description

This operation gets the state of the `_release` flag of the sequence.

Parameters

*in void *sequence* - a pointer to the DDS_<sequence-name>.

Return Value

DDS_boolean - the present state of the `_release` flag of the sequence.

Detailed Description

This operation gets the present state of the `_release` flag of the sequence. If the flag returned is `TRUE`, the sequence effectively “owns” the resource pointed to by `_buffer`; if the flag returned is `FALSE`, the application is responsible for the resource. If, for example, a sequence is returned from an operation with its release flag set to `FALSE`, calling `DDS_free` on the returned sequence pointer does not deallocate the memory pointed to by `_buffer`. Before calling `DDS_free` on the `_buffer` member of a sequence directly, the application should check the `_release` flag using `DDS_sequence_get_release`. If it returns `FALSE`, the application should not invoke `DDS_free` on the `_buffer` member; doing so produces undefined behaviour. Passing a `DDS_OBJECT_NIL` pointer or a pointer to something other than a sequence type to `DDS_sequence_get_release` produces undefined behaviour.

1.3.5.2.2 DDS_<sequence-name>__alloc

Synopsis

```
#include <dds_dcps.h>
DDS_<sequence-name>
    DDS_<sequence-name>__alloc
        (void);
```

Description

This operation allocates a new DDS_<sequence-name>.

Parameters

<none>

Return Value

DDS_<sequence-name> - the pointer to the newly created empty *DDS_<sequence-name>*. In case of an error, a *DDS_OBJECT_NIL* pointer is returned.

Detailed Description

This operation allocates a new empty *DDS_<sequence-name>*. This operation does not allocate the buffer and leave the sequence empty by setting the *_length* and *_maximum* attributes to zero and the *_buffer* attribute to *DDS_OBJECT_NIL*. The application may also allocate the *DDS_<sequence-name>* as a variable on stack. In this case the application may not use *DDS_free* on the sequence. In case the *DDS_<sequence-name>* was allocated by this operation, and the application wants to release the *DDS_<sequence-name>* it must be released using *DDS_free* on the sequence.

In case there are insufficient resources available to allocate the *DDS_<sequence-name>*, a *DDS_OBJECT_NIL* pointer is returned instead.

1.3.5.2.3 DDS_<sequence-element-type>_allocbuf**Synopsis**

```
#include <dds_dcps.h>
    DDS_<sequence-element-type> *
        DDS_<sequence-name>_allocbuf
            (DDS_unsigned_long len);
```

Description

This operation allocates a new *DDS_<sequence-element-type>* buffer.

Parameters

<none>

Return Value

DDS_<sequence-element-type> - the pointer to the newly created buffer of *DDS_<sequence-element-type>*. In case of an error, a *DDS_OBJECT_NIL* pointer is returned.

Detailed Description

This operation allocates a new buffer of `DDS_<sequence-element-type>`. The application may also allocate the buffer of `DDS_<sequence-element-type>` as a variable on stack. In this case the application may not use `DDS_free` on the buffer. Furthermore, the application may only use `DDS_free` on the sequence when the `_release` flag of the sequence is set to `FALSE` and/or the `_buffer` pointer is set to `DDS_OBJECT_NIL` to prevent the buffer from being released. In case the buffer of `DDS_<sequence-element-type>` was allocated by this operation, and the application wants to release the buffer of `DDS_<sequence-element-type>` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the buffer of `DDS_<sequence-element-type>`, a `DDS_OBJECT_NIL` pointer is returned instead.

1.3.5.2.4 `DDS_<QosPolicy>__alloc`

Synopsis

```
#include <dds_dcps.h>
DDS_<QosPolicy>
    DDS_<QosPolicy>__alloc
        (void);
```

Description

This operation allocates a new `DDS_<QosPolicy>`.

Parameters

<none>

Return Value

`DDS_<QosPolicy>` - the handle to the newly created `DDS_<QosPolicy>`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation allocates a new `DDS_<QosPolicy>`. The behaviour is identical to `DDS_<sequence-name>__alloc` except that it creates a `QosPolicy` structure including its embedded sequences. Further, the embedded buffers are not allocated.

1.3.5.2.5 `DDS_<Status>__alloc`

Synopsis

```
#include <dds_dcps.h>
DDS_<Status>
```

```
DDS_<Status>__alloc
(void);
```

Description

This operation allocates a new DDS_<Status>.

Parameters

<none>

Return Value

DDS_<Status> - the handle to the newly created DDS_<Status>. In case of an error, a DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation allocates a new DDS_<Status>. The behaviour is identical to DDS_<sequence-name>__alloc except that it creates a Status structure including its embedded sequences. Further, the embedded buffers are not allocated.

1.3.5.2.6 DDS_string_alloc

Synopsis

```
#include <dds_dcps.h>
DDS_char *
    DDS_string_alloc
        (DDS_unsigned_long len);
```

Description

This operation dynamically allocates a string of a specified length.

Parameters

in DDS_unsigned_long len - the length of the string to allocate. The allocated string has length len+1 (1 character is allocated extra for the terminating NUL character).

Return Value

DDS_char * - the pointer to the allocated string. If there are insufficient resources available, a DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation dynamically allocates a string of a specified length. The allocated string has length `len+1` (1 character is allocated extra for the terminating `'0'` character). If there are insufficient resources available, a `DDS_OBJECT_NIL` pointer is returned.

A string that is allocated via `DDS_string_alloc` must be freed using the operation `DDS_free`.

1.3.5.2.7 DDS_free

Synopsis

```
#include <dds_dcps.h>
void
    DDS_free
        (void *);
```

Description

This operation releases the allocated resources for the object in the parameter.

Parameters

in `void *` - contains the object which resources should be released.

Return Value

<none>

Detailed Description

This operation releases the allocated resources for the object in the parameter. The parameter may be a sequence in which case both the sequence and the sequence buffer are released since this operation operates recursively. Or the parameter may be a sequence buffer in case only the buffer is released. In both cases, the application is responsible to call this operation on the proper object in order to release the resources.

This operation may only be used when the resource was allocated using one of the `_alloc` operations. In case the object was declared as a variable on stack, the application may not use `DDS_free` on this object.

This means that there are four combinations of allocation possible:

Both the sequence and the buffer is allocated using the `DDS_<sequence-name>__alloc` and `DDS_<sequence-name>_allocbuf` operation. In this case the `DDS_free` operation must be used on the sequence to release both.

- The sequence is allocated on stack and the buffer is allocated using the `DDS_<sequence-name>_allocbuf` operation. In this case the sequence may not be released using the `DDS_free` operation but the buffer must be released using the `DDS_free` operation (operated on the buffer).
- The sequence is allocated using the `DDS_<sequence-name>__alloc` operation and the buffer is allocated on stack. In this case the `DDS_free` operation must be used on the sequence but the buffer may not be released using the `DDS_free` operation. Since the `DDS_free` operation works recursively, the application must put the `_release` flag of the sequence to `FALSE` and/or the `_buffer` pointer to `DDS_OBJECT_NIL` to prevent the buffer from being released.
- Both the sequence and the buffer are allocated on stack. In this case the `DDS_free` operation may not be used.

1.4 Listeners Interfaces

The `Listener` provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a `DDS_QosPolicy` setting, etc. The `Listener` is related to changes in communication status.

The `Listener` interfaces are designed as an interface at PIM level. In other words, such an interface is part of the application which must implement the interface operations. These operations must be provided by the application. **All** `Listener` operations **must** be implemented, it is up to the application whether an operation is empty or contains some functionality.

Each DCPS `DDS_Entity` supports its own specialized kind of `Listener`. Therefore, the following `Listeners` are available:

- `DDS_DomainParticipantListener`
- `DDS_TopicListener`
- `DDS_PublisherListener`
- `DDS_DataWriterListener`
- `DDS_SubscriberListener`
- `DDS_DataReaderListener`

For example, since a `DDS_DataReader` is a kind of `DDS_Entity`, it has the ability to have a `Listener` associated with it. In this case, the associated `Listener` must be of type `DDS_DataReaderListener`. This interface must be implemented by the application. *All* `DDS_DataReaderListener` operations **must** be implemented, it is up to the application whether an operation is empty or contains some functionality.

As an example, one of the operations in the `DDS_DataReaderListener` is the `DDS_DataReaderListener_on_liveliness_changed`. This operation (implemented by the application) will be called by the Data Distribution Service when the liveliness of the associated `DDS_DataWriter` has changed. In other words, it serves as a callback function to the event of a change in liveliness. The parameters of the operation are supplied by the Data Distribution Service. In this example, the pointer to the `DDS_DataReader` and the status of the liveliness are provided.

Implementation

The struct `DDS_<Entity>Listener` represents the implementation of the Listener for an `<Entity>`. Since a Listener is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The Listener is allocated using the appropriate `DDS_<Entity>Listener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application (when the status is enabled). It is up to the application whether an operation is empty or contains some functionality. An example is presented of the allocation and initialization of a `DDS_DataReaderListener` which is only enabled for the `on_liveliness_changed`. The `on_liveliness_changed` operation is provided by the application:

```
#include "dds_dcps.h"
static struct DDS_DataReaderListener msgListener;
DDS_FooDataReader FooDR;
/* at this point, it is not important how to create the FooDR
*/
DataWriterListenerData UserDefined_ListenerData;
/* at this point, it is not important how
UserDefined_ListenerData is implemented.
This parameter can be used for Listener identification.
If not used, the parameter may be NULL. */
/* Prepare a listener for the Foo DataReader. */
msgListener = DDS_DataReaderListener__alloc();
msgListener.listener_data = UserDefined_ListenerData;
msgListener.on_requested_deadline_missed = NULL;
msgListener.on_requested_incompatible_qos = NULL;
msgListener.on_sample_rejected = NULL;
msgListener.on_liveliness_changed =
    (void (*)(void *, DDS_DataReader)) on_live_change;
msgListener.on_data_available = NULL;
msgListener.on_subscription_matched = NULL;
msgListener.on_sample_lost = NULL;

/* Set the Listener with a mask only
```

```

        to trigger on on_liveliness_changed. */
    status = DDS_DataReader_set_listener
        (FoodR,
         &msgListener,
         DDS_LIVELINESS_CHANGED_STATUS);

```

1.4.1 Struct DDS_<Entity>Listener

The struct DDS_<Entity>Listener represents the implementation of a Listener.

The interface description applies to the different types of <Entity>, that is the DomainParticipant, Topic, Publisher, DataWriter, Subscriber or DataReader. The actual attributes depends on the <Entity>. Only for the DomainParticipant all the fields are applicable. the description of these structs is as follows:

```

typedef struct DDS_DomainParticipantListener
    *DDS_DomainParticipantListener;
struct DDS_DomainParticipantListener
{
    void *listener_data;
    DDS_DomainParticipantListener_InconsistentTopicListener
        on_inconsistent_topic;

    DDS_DomainParticipantListener_OfferedDeadlineMissedListener
        on_offered_deadline_missed;

    DDS_DomainParticipantListener_OfferedIncompatibleQosListener
        on_offered_incompatible_qos;
    DDS_DomainParticipantListener_LivelinessLostListener
        on_liveliness_lost;
    DDS_DomainParticipantListener_PublicationMatchListener
        on_publication_matched;

    DDS_DomainParticipantListener_RequestedDeadlineMissedListener
        on_requested_deadline_missed;
    DDS_DomainParticipantListener_
        RequestedIncompatibleQosListener
        on_requested_incompatible_qos;
    DDS_DomainParticipantListener_SampleRejectedListener
        on_sample_rejected;
    DDS_DomainParticipantListener_LivelinessChangedListener
        on_liveliness_changed;
    DDS_DomainParticipantListener_DataAvailableListener
        on_data_available;
    DDS_DomainParticipantListener_SubscriptionMatchListener

```

```

        on_subscription_matched;
    DDS_DomainParticipantListener_SampleLostListener
        on_sample_lost;
    DDS_DomainParticipantListener_DataOnReadersListener
        on_data_on_readers;
};
typedef struct DDS_TopicListener
    *DDS_TopicListener;
struct DDS_TopicListener
{
    void *listener_data;
    DDS_TopicListener_InconsistentTopicListener
        on_inconsistent_topic;
};
typedef struct DDS_PublisherListener
    *DDS_PublisherListener;
struct DDS_PublisherListener
{
    void *listener_data;
    DDS_PublisherListener_OfferedDeadlineMissedListener
        on_offered_deadline_missed;
    DDS_PublisherListener_OfferedIncompatibleQosListener
        on_offered_incompatible_qos;
    DDS_PublisherListener_LivelinessLostListener
        on_liveliness_lost;
    DDS_PublisherListener_PublicationMatchListener
        on_publication_matched;
};
typedef struct DDS_DataWriterListener
    *DDS_DataWriterListener;
struct DDS_DataWriterListener
{
    void *listener_data;
    DDS_DataWriterListener_OfferedDeadlineMissedListener
        on_offered_deadline_missed;
    DDS_DataWriterListener_OfferedIncompatibleQosListener
        on_offered_incompatible_qos;
    DDS_DataWriterListener_LivelinessLostListener
        on_liveliness_lost;
    DDS_DataWriterListener_PublicationMatchListener
        on_publication_matched;
};
typedef struct DDS_SubscriberListener
    *DDS_SubscriberListener;
struct DDS_SubscriberListener

```

```

{
    void *listener_data;
    DDS_SubscriberListener_RequestedDeadlineMissedListener
        on_requested_deadline_missed;
    DDS_SubscriberListener_RequestedIncompatibleQosListener
        on_requested_incompatible_qos;
    DDS_SubscriberListener_SampleRejectedListener
        on_sample_rejected;
    DDS_SubscriberListener_LivelinessChangedListener
        on_liveliness_changed;
    DDS_SubscriberListener_DataAvailableListener
        on_data_available;
    DDS_SubscriberListener_SubscriptionMatchListener
        on_subscription_matched;
    DDS_SubscriberListener_SampleLostListener
        on_sample_lost;
    DDS_SubscriberListener_DataOnReadersListener
        on_data_on_readers;
};
typedef struct DDS_DataReaderListener
    *DDS_DataReaderListener;
struct DDS_DataReaderListener
{
    void *listener_data;
    DDS_DataReaderListener_RequestedDeadlineMissedListener
        on_requested_deadline_missed;
    DDS_DataReaderListener_RequestedIncompatibleQosListener
        on_requested_incompatible_qos;
    DDS_DataReaderListener_SampleRejectedListener
        on_sample_rejected;
    DDS_DataReaderListener_LivelinessChangedListener
        on_liveliness_changed;
    DDS_DataReaderListener_DataAvailableListener
        on_data_available;
    DDS_DataReaderListener_SubscriptionMatchListener
        on_subscription_matched;
    DDS_DataReaderListener_SampleLostListener
        on_sample_lost;
};
/* implemented API operations
 *      <no operations> */

```

The next paragraphs describes the usage of the DDS_<Entity>Listener structs.

1.4.2 DDS_DomainParticipantListener

Synopsis

```
#include <dds_dcps.h>
struct DDS_DomainParticipantListener
{
    void *listener_data;
    DDS_DomainParticipantListener_InconsistentTopicListener
        on_inconsistent_topic;
    DDS_DomainParticipantListener_OfferedDeadlineMissedListener
        on_offered_deadline_missed;
    DDS_DomainParticipantListener_OfferedIncompatibleQosListener
        on_offered_incompatible_qos;
    DDS_DomainParticipantListener_LivelinessLostListener
        on_liveliness_lost;
    DDS_DomainParticipantListener_PublicationMatchListener
        on_publication_matched;
    DDS_DomainParticipantListener_RequestedDeadlineMissedListener
        on_requested_deadline_missed;
    DDS_DomainParticipantListener_RequestedIncompatibleQosListener
        on_requested_incompatible_qos;
    DDS_DomainParticipantListener_SampleRejectedListener
        on_sample_rejected;
    DDS_DomainParticipantListener_LivelinessChangedListener
        on_liveliness_changed;
    DDS_DomainParticipantListener_DataAvailableListener
        on_data_available;
    DDS_DomainParticipantListener_SubscriptionMatchListener
        on_subscription_matched;
    DDS_DomainParticipantListener_SampleLostListener
        on_sample_lost;
    DDS_DomainParticipantListener_DataOnReadersListener
        on_data_on_readers;
};
```

Description

The struct `DDS_DomainParticipantListener` represents the implementation of the `DomainParticipantListener`.

Attributes

*void *listener_data* - a pointer to a user-defined object, which may be used for identification of the Listener.

DDS_DomainParticipantListener_InconsistentTopicListener on_inconsistent_topic - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_OfferedDeadlineMissedListener
on_offered_deadline_missed - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_OfferedIncompatibleQosListener
on_offered_incompatible_qos - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_LivelinessLostListener
on_liveliness_lost - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_PublicationMatchListener
on_publication_matched - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_RequestedDeadlineMissedListener
on_requested_deadline_missed - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_RequestedIncompatibleQosListener
on_requested_incompatible_qos - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_SampleRejectedListener
on_sample_rejected - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_LivelinessChangedListener
on_liveliness_changed - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_DataAvailableListener
on_data_available - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_SubscriptionMatchListener
on_subscription_matched - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_SampleLostListener
on_sample_lost - a pointer to the call back function implemented by the application.

DDS_DomainParticipantListener_DataOnReadersListener
on_data_on_readers - a pointer to the call back function implemented by the application.

Detailed Description

The struct `DDS_DomainParticipantListener` represents the implementation of the `Listener` for the `DomainParticipant`. Since a `Listener` is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The `Listener` is allocated using the `DDS_DomainParticipantListener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application. It is up to the application whether an operation is empty or contains some functionality. The `listener_data` attribute is a pointer to an application defined object. This attribute can be used to supply the identity of the `Listener`, which has been called. A description of the other attributes is given in the appropriate `on_<status>` callback operations in each `Listener`.

1.4.3 DDS_TopicListener

Synopsis

```
#include <dds_dcps.h>
struct DDS_TopicListener
{
    void *listener_data;
    DDS_TopicListener_InconsistentTopicListener
        on_inconsistent_topic;
};
```

Description

The struct `DDS_TopicListener` represents the implementation of the `TopicListener`.

Attributes

`void *listener_data` - a pointer to a user-defined object, which may be used for identification of the `Listener`.

`DDS_TopicListener_InconsistentTopicListener`
`on_inconsistent_topic` - a pointer to the call back function implemented by the application.

Detailed Description

The struct `DDS_TopicListener` represents the implementation of the `Listener` for the `Topic`. Since a `Listener` is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The `Listener` is allocated using the `DDS_TopicListener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application. It is up to the application whether an operation is empty or contains some functionality. The

`listener_data` attribute is a pointer to an application defined object. This attribute can be used to supply the identity of the `Listener`, which has been called. A description of the other attributes is given in the appropriate `on_<status>` callback operations in each `Listener`.

1.4.4 DDS_PublisherListener

Synopsis

```
#include <dds_dcps.h>
struct DDS_publisherListener
{
    void *listener_data;
    DDS_publisherListener_OfferedDeadlineMissedListener
        on_offered_deadline_missed;
    DDS_publisherListener_OfferedIncompatibleQosListener
        on_offered_incompatible_qos;
    DDS_publisherListener_LivelinessLostListener
        on_liveliness_lost;
    DDS_publisherListener_PublicationMatchListener
        on_publication_matched;
};
```

Description

The struct `DDS_publisherListener` represents the implementation of the `publisherListener`.

Attributes

`void *listener_data` - a pointer to a user-defined object, which may be used for identification of the `Listener`.

`DDS_publisherListener_OfferedDeadlineMissedListener`
`on_offered_deadline_missed` - a pointer to the call back function implemented by the application.

`DDS_publisherListener_OfferedIncompatibleQosListener`
`on_offered_incompatible_qos` - a pointer to the call back function implemented by the application.

`DDS_publisherListener_LivelinessLostListener`
`on_liveliness_lost` - a pointer to the call back function implemented by the application.

`DDS_publisherListener_PublicationMatchListener`
`on_publication_matched` - a pointer to the call back function implemented by the application.

Detailed Description

The struct `DDS_publisherListener` represents the implementation of the `Listener` for the publisher. Since a `Listener` is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The `Listener` is allocated using the `DDS_publisherListener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application. It is up to the application whether an operation is empty or contains some functionality. The `listener_data` attribute is a pointer to an application defined object. This attribute can be used to supply the identity of the `Listener`, which has been called. A description of the other attributes is given in the appropriate `on_<status>` callback operations in each `Listener`.

1.4.5 DDS_DataWriterListener

Synopsis

```
#include <dds_dcps.h>
struct DDS_DataWriterListener
{
    DDS_DataWriterListener_OfferedDeadlineMissedListener
        on_offered_deadline_missed;
    DDS_DataWriterListener_OfferedIncompatibleQosListener
        on_offered_incompatible_qos;
    DDS_DataWriterListener_LivelinessLostListener
        on_liveliness_lost;
    DDS_DataWriterListener_PublicationMatchListener
        on_publication_matched;
};
```

Description

The struct `DDS_DataWriterListener` represents the implementation of the `DataWriterListener`.

Attributes

`void *listener_data` - a pointer to a user-defined object, which may be used for identification of the `Listener`.

`DDS_DataWriterListener_OfferedDeadlineMissedListener`
`on_offered_deadline_missed` - a pointer to the call back function implemented by the application.

`DDS_DataWriterListener_OfferedIncompatibleQosListener`
`on_offered_incompatible_qos` - a pointer to the call back function implemented by the application.

DDS_DataWriterListener_LivelinessLostListener
on_liveliness_lost - a pointer to the call back function implemented by the application.

DDS_DataWriterListener_PublicationMatchListener
on_publication_matched - a pointer to the call back function implemented by the application.

Detailed Description

The struct `DDS_DataWriterListener` represents the implementation of the Listener for the DataWriter. Since a Listener is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The Listener is allocated using the `DDS_DataWriterListener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application. It is up to the application whether an operation is empty or contains some functionality. The `listener_data` attribute is a pointer to an application defined object. This attribute can be used to supply the identity of the Listener, which has been called. A description of the other attributes is given in the appropriate `on_<status>` callback operations in each Listener.

1.4.6 DDS_SubscriberListener

Synopsis

```
#include <dds_dcps.h>
struct DDS_SubscriberListener
{
    void *listener_data;
    DDS_SubscriberListener_RequestedDeadlineMissedListener
        on_requested_deadline_missed;
    DDS_SubscriberListener_RequestedIncompatibleQosListener
        on_requested_incompatible_qos;
    DDS_SubscriberListener_SampleRejectedListener
        on_sample_rejected;
    DDS_SubscriberListener_LivelinessChangedListener
        on_liveliness_changed;
    DDS_SubscriberListener_DataAvailableListener
        on_data_available;
    DDS_SubscriberListener_SubscriptionMatchListener
        on_subscription_matched;
    DDS_SubscriberListener_SampleLostListener
        on_sample_lost;
    DDS_SubscriberListener_DataOnReadersListener
        on_data_on_readers;
};
```

Description

The struct `DDS_SubscriberListener` represents the implementation of the `SubscriberListener`.

Attributes

`void *listener_data` - a pointer to a user-defined object, which may be used for identification of the `Listener`.

`DDS_SubscriberListener_RequestedDeadlineMissedListener`
`on_requested_deadline_missed` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_RequestedIncompatibleQosListener`
`on_requested_incompatible_qos` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_SampleRejectedListener`
`on_sample_rejected` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_LivelinessChangedListener`
`on_liveliness_changed` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_DataAvailableListener`
`on_data_available` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_SubscriptionMatchListener`
`on_subscription_matched` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_SampleLostListener` `on_sample_lost` - a pointer to the call back function implemented by the application.

`DDS_SubscriberListener_DataOnReadersListener`
`on_data_on_readers` - a pointer to the call back function implemented by the application.

Detailed Description

The struct `DDS_SubscriberListener` represents the implementation of the `Listener` for the `Subscriber`. Since a `Listener` is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The `Listener` is allocated using the `DDS_SubscriberListener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application. It is up to the application whether an operation is empty or contains some functionality. The `listener_data` attribute is a pointer to an application

defined object. This attribute can be used to supply the identity of the `Listener`, which has been called. A description of the other attributes is given in the appropriate `on_<status>` callback operations in each `Listener`.

1.4.7 DDS_DataReaderListener

Synopsis

```
#include <dds_dcps.h>
struct DDS_DataReaderListener
{
    void *listener_data;
    DDS_DataReaderListener_RequestedDeadlineMissedListener
        on_requested_deadline_missed;
    DDS_DataReaderListener_RequestedIncompatibleQosListener
        on_requested_incompatible_qos;
    DDS_DataReaderListener_SampleRejectedListener
        on_sample_rejected;
    DDS_DataReaderListener_LivelinessChangedListener
        on_liveliness_changed;
    DDS_DataReaderListener_DataAvailableListener
        on_data_available;
    DDS_DataReaderListener_SubscriptionMatchListener
        on_subscription_matched;
    DDS_DataReaderListener_SampleLostListener
        on_sample_lost;
};
```

Description

The struct `DDS_DataReaderListener` represents the implementation of the `DataReaderListener`.

Attributes

`void *listener_data` - a pointer to a user-defined object, which may be used for identification of the `Listener`.

`DDS_DataReaderListener_RequestedDeadlineMissedListener`
`on_requested_deadline_missed` - a pointer to the call back function implemented by the application.

`DDS_DataReaderListener_RequestedIncompatibleQosListener`
`on_requested_incompatible_qos` - a pointer to the call back function implemented by the application.

`DDS_DataReaderListener_SampleRejectedListener`
`on_sample_rejected` - a pointer to the call back function implemented by the application.

DDS_DataReaderListener_LivelinessChangedListener
on_liveliness_changed - a pointer to the call back function implemented by the application.

DDS_DataReaderListener_DataAvailableListener
on_data_available - a pointer to the call back function implemented by the application.

DDS_DataReaderListener_SubscriptionMatchListener
on_subscription_matched - a pointer to the call back function implemented by the application.

DDS_DataReaderListener_SampleLostListener *on_sample_lost* - a pointer to the call back function implemented by the application.

Detailed Description

The struct `DDS_DataReaderListener` represents the implementation of the `Listener` for the `DataReader`. Since a `Listener` is implemented as a struct of pointers, the application must allocate this struct and initialise these pointers. The `Listener` is allocated using the `DDS_DataReaderListener__alloc` operation. Each pointer must point to the appropriate callback operation defined in the application. It is up to the application whether an operation is empty or contains some functionality. The `listener_data` attribute is a pointer to an application defined object. This attribute can be used to supply the identity of the `Listener`, which has been called. A description of the other attributes is given in the appropriate `on_<status>` callback operations in each `Listener`.

1.5 Inheritance of Abstract Operations

The information provided here conforms to the

- PIM part of the DDS-DCPS specification (for module descriptions)
- PSM part of the DDS-DCPS specification (for class and operation descriptions).

For detailed information refer to the *OMG C Language Mapping Specification*.

At PIM level, inheritance is used to define abstract classes and operations. The OMG IDL PSM defines the interface for an application to interact with the Data Distribution Service. The DCPS API for the C programming language conforms to the IDL to C mapping as specified in the *OMG C Language Mapping Specification*.

Inheritance of operations is not implemented when different type parameters for the same operation are used. In this case operations are implemented in their respective derived class (e.g. `DDS_<Entity>_get_qos` and `DDS_<Entity>_set_qos`). These operations are commented out in the IDL PSM.

CHAPTER

2 DCPS Modules

DCPS is divided into five modules, which are described briefly in this chapter. Each module consists of several classes as defined at PIM level in the DDS-DCPS specification. Some of the classes as described in the PIM are implemented as a struct in the PSM; these classes are treated as a class in this chapter according to the PIM with a remark about their implementation (struct). In the next chapter their actual implementations are described.

Each class contains several operations, which may be abstract. Those classes, which are implemented as a struct do not have any operations. The modules and the classes are ordered conform the DDS-DCPS specification. The classes, interfaces, structs and operations are described in the next chapter.

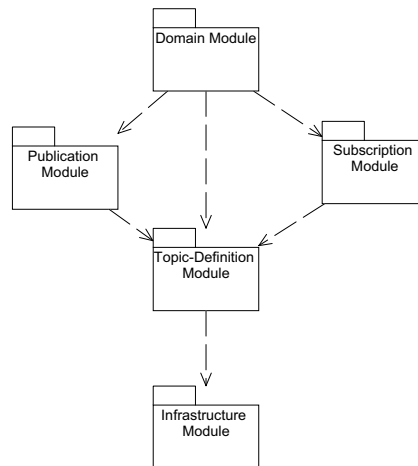


Figure 2 DCPS Module Composition

2.1 Functionality

The modules have the following function in the Data Distribution Service:

- **Infrastructure Module:** This module defines the abstract classes and interfaces, which are refined by the other modules. It also provides the support for the interaction between the application and the Data Distribution Service (state-based and event-based)

- **Domain Module** - This module contains the `DDS_DomainParticipant` class, which is the entry point of the application, the `DDS_DomainParticipantFactory` class and `DDS_DomainParticipantListener` interface
- **Topic-Definition Module** - This module contains the `DDS_Topic`, `DDS_ContentFilteredTopic` and `DDS_MultiTopic` classes. It also contains the `DDS_TopicListener` interface and all support to define `DDS_Topic` objects and assign `QosPolicy` settings to them
- **Publication Module** - This module contains the `DDS_Publisher` and `DDS_DataWriter` classes. It also contains the `DDS_PublisherListener` and `DDS_DataWriterListener` interfaces
- **Subscription Module** - This module contains the `DDS_Subscriber`, `DDS_DataReader`, `DDS_ReadCondition` and `DDS_QueryCondition` classes. It also contains the `DDS_SubscriberListener` and `DDS_DataReaderListener` interfaces

2.2 Infrastructure Module

This module defines the abstract classes and interfaces, which, in the PIM definition, are refined by the other modules. It also provides the support for the interaction between the application and the Data Distribution Service (event-based and state-based). The event-based interaction is supported by `DDS_Listeners`, the state-based interaction is supported by `DDS_WaitSets` and `DDS_Conditions`.

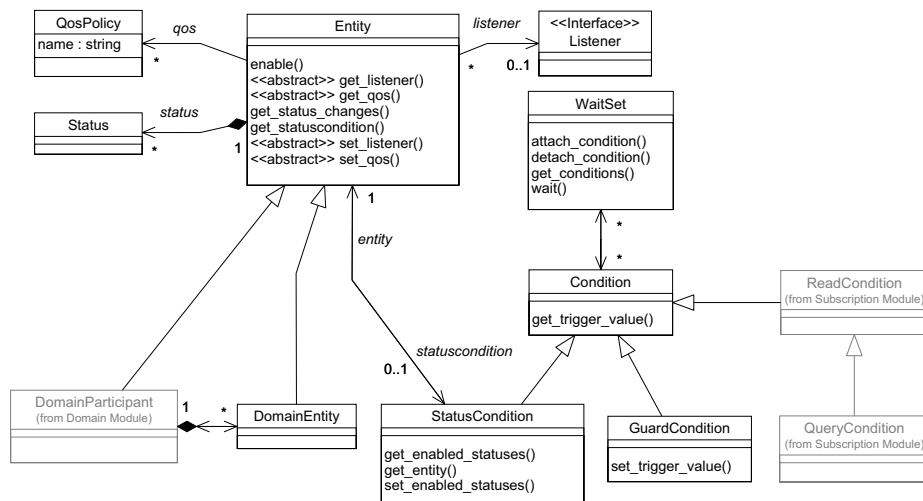


Figure 3 DCPS Infrastructure Module's Class Model

This module contains the following classes:

- `DDS_Entity` (abstract)
- `DDS_DomainEntity` (abstract)
- `DDS_QosPolicy` (abstract, struct)
- `DDS_Listener` (interface)
- `DDS_Status` (abstract, struct)
- `DDS_WaitSet`
- `DDS_Condition`
- `DDS_GuardCondition`
- `DDS_StatusCondition`

2.3 Domain Module

This module contains the class `DDS_DomainParticipant`, which acts as an entry point of the Data Distribution Service and acts as a factory for many of the classes. The `DDS_DomainParticipant` also acts as a container for the other objects that make up the Data Distribution Service. It isolates applications within the same `Domain` from other applications in a different `Domain` on the same set of computers. A `Domain` is a “virtual network” and applications with the same `domainId` are isolated from applications with a different `domainId`. In this way, several independent distributed applications can coexist in the same physical network without interfering, or even being aware of each other.

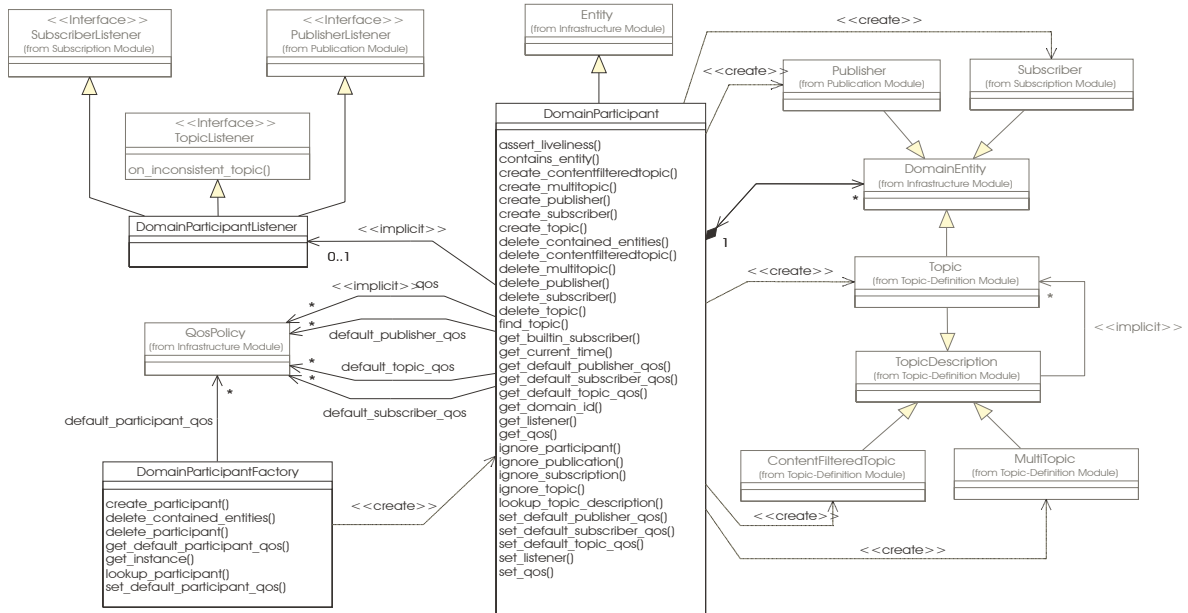


Figure 4 DCPS Domain Module's Class Model

This module contains the following classes:

- DDS_DomainParticipant
- DDS_DomainParticipantFactory
- DDS_DomainParticipantListener (interface)
- DDS_Domain (*not depicted*)

2.4 Topic-Definition Module

This module contains the `DDS_Topic`, `DDS_ContentFilteredTopic` and `DDS_MultiTopic` classes. It also contains the `DDS_TopicListener` interface and all support to define `DDS_Topic` objects and assign `QosPolicy` settings to them.

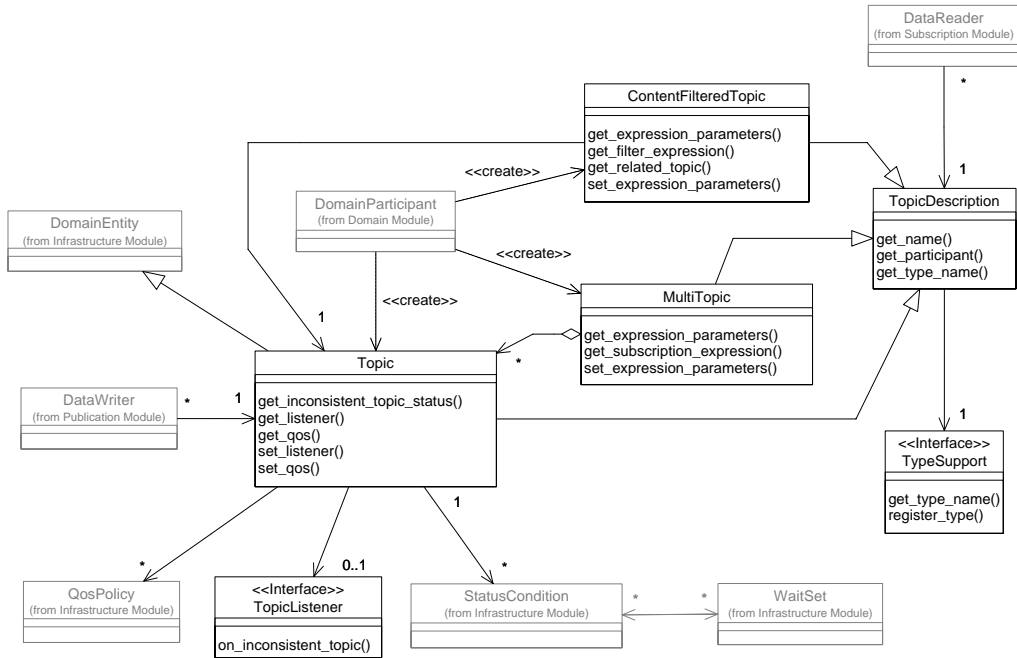


Figure 5 DCPS Topic-Definition Module’s Class Model

This module contains the following classes:

- DDS_TopicDescription (abstract)
- DDS_Topic
- DDS_ContentFilteredTopic
- DDS_MultiTopic
- DDS_TopicListener (interface)
- Topic-Definition type specific classes

Topic-Definition type specific classes contain the generic class and the generated data type specific classes. In case of the user-defined data type Foo (this also applies to other types), defined in the module SPACE; “Topic-Definition type specific classes” contains the following classes:

- DDS_TypeSupport (abstract)
- SPACE_FooTypeSupport

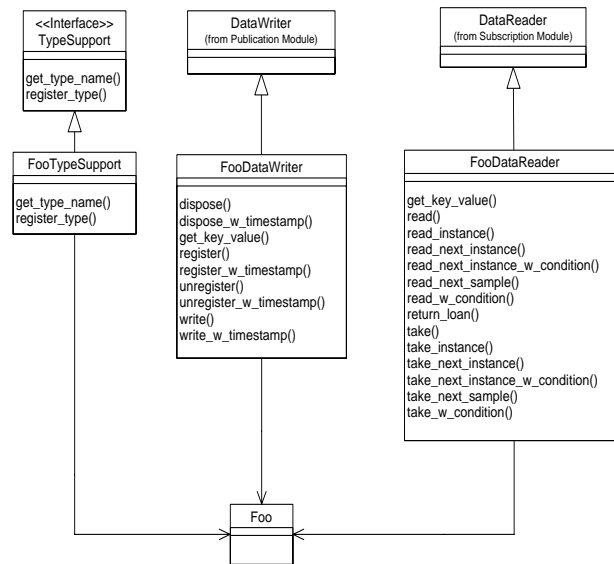


Figure 6 Data Type “Foo” Typed Classes Pre-processor Generation

2.5 Publication Module

This module supports writing of the data, it contains the `DDS_Publisher` and `DDS_DataWriter` classes. It also contains the `DDS_PublisherListener` and `DDS_DataWriterListener` interfaces. Furthermore, it contains all support needed for publication.

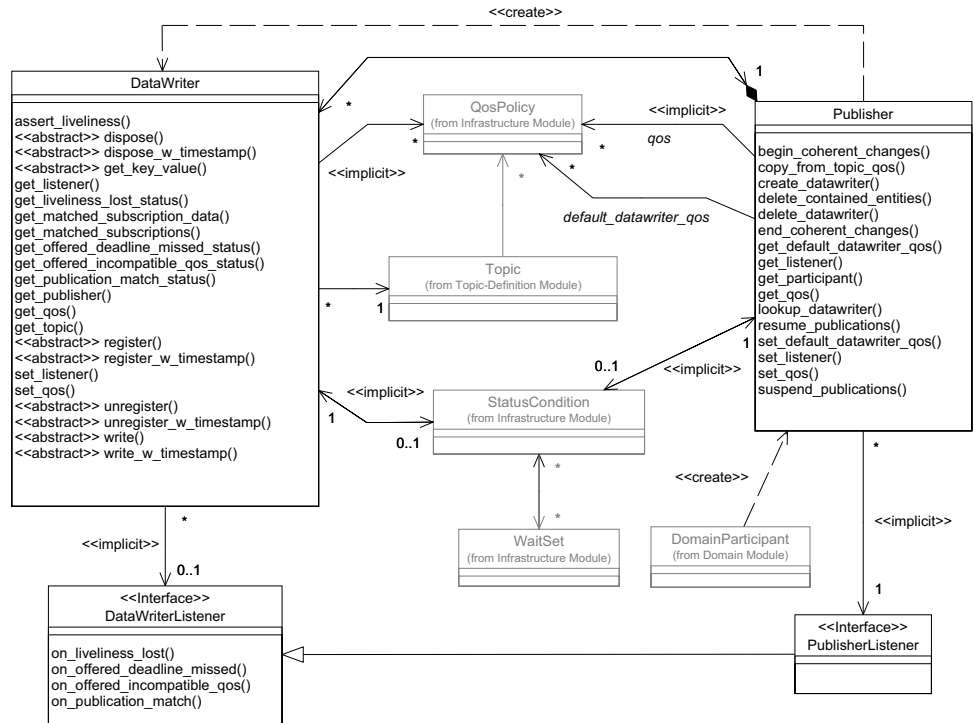


Figure 7 DCPS Publication Module's Class Model

This module contains the following classes:

- DDS_Publisher
- DDS_PublisherListener (interface)
- DDS_DataWriterListener (interface)
- Publication type specific classes

Publication type specific classes contain the generic class and the generated data type specific classes. In case of the user-defined data type Foo (this also applies to other types), defined in the module SPACE; "Publication type specific classes" contains the following classes:

- DDS_DataWriter (abstract)
- SPACE_FooDataWriter

2.6 Subscription Module

This module supports access to the data, it contains the `DDS_Subscriber`, `DDS_DataReader`, `DDS_ReadCondition` and `DDS_QueryCondition` classes. It also contains the `DDS_SubscriberListener` and `DDS_DataReaderListener` interfaces. Furthermore, it contains all support needed for subscription.

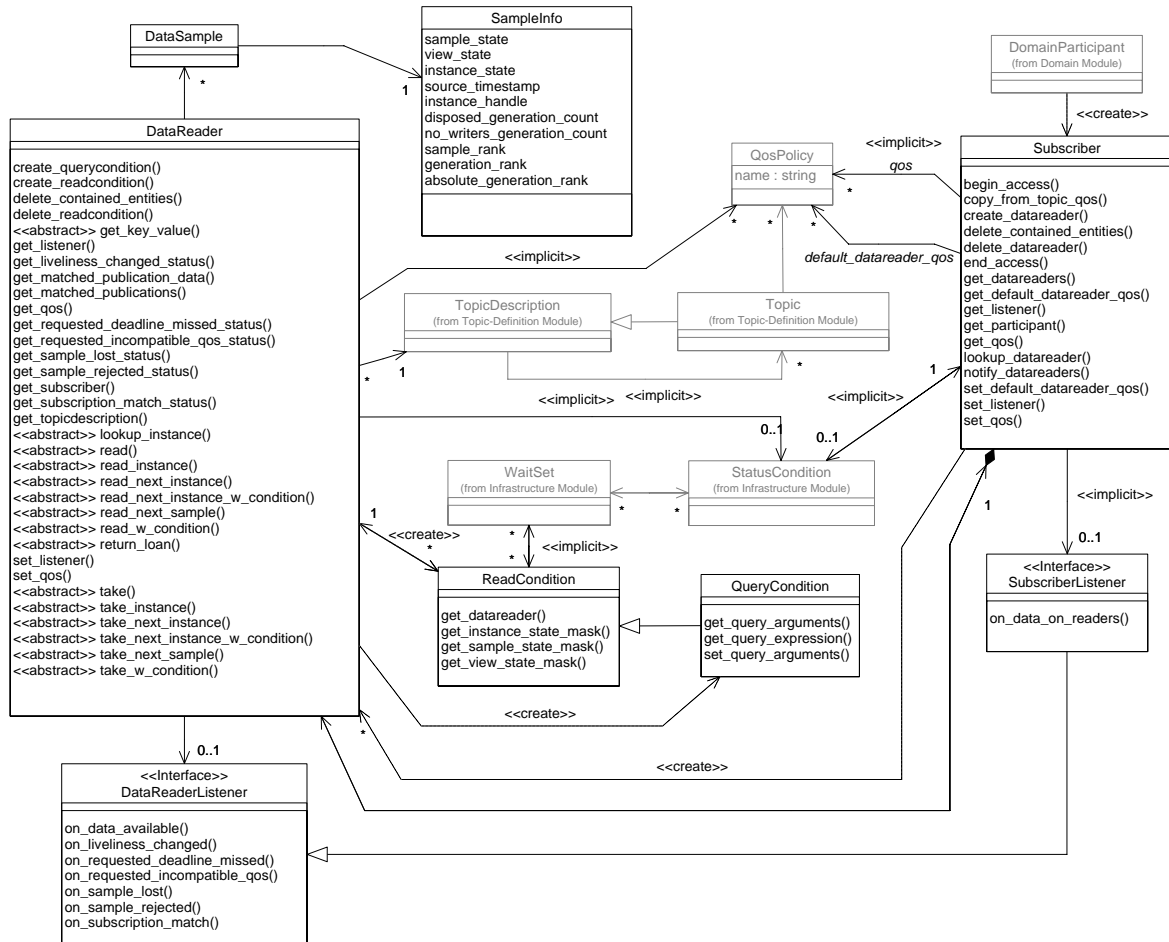


Figure 8 DCPS Subscription Module's Class Model

This module contains the following classes:

- `DDS_Subscriber`
- `DDS_DataSample`
- `DDS_SampleInfo` (struct)

- `DDS_SubscriberListener` (interface)
- `DDS_DataReaderListener` (interface)
- `DDS_ReadCondition`
- `DDS_QueryCondition`
- Subscription type specific classes

Subscription type specific classes contain the generic class and the generated data type specific classes. In case of the user-defined data type `Foo` (this also applies to other types), defined in the module `SPACE`; “Subscription type specific classes” contains the following classes:

- `DDS_DataReader` (abstract)
- `SPACE_FooDataReader`

3 DCPS Classes and Operations

This chapter describes, for each module, its classes and operations in detail. Each module consists of several classes as defined at PIM level in the DDS-DCPS specification. Some of the classes are implemented as a struct in the PSM. Some of the other classes are abstract, which means they contain some abstract operations.

*The Listener interfaces are designed as an interface at PIM level. In other words, the application must implement the interface operations. Therefore, all Listener classes are abstract. A user-defined class for these operations must be provided by the application which must extend from the **specific** Listener class. **All** Listener operations **must** be implemented in the user-defined class. It is up to the application whether an operation is empty or contains some functionality.*

The Listener interfaces in the C API are implemented as structs containing function pointers. All the function pointer attributes within the struct must be assigned to a function. It is up to the application whether a function is empty or contains some functionality.

Each class contains several operations, which may be abstract (base class). Abstract operations are not implemented in their base class, but in a type specific class or an application defined class (in case of a Listener). Classes that are implemented as a struct do not have any operations. Some operations are inherited, which means they are implemented in their base class.

The abstract operations in a class are listed (including their synopsis), but not implemented in that class. These operations are implemented in their respective derived classes. The interfaces are fully described, since they must be implemented by the application.

i

General note for type Space: The name `Space.h` is derived from the IDL file `Space.idl`, that defines `Foo`.

3.1 Infrastructure Module

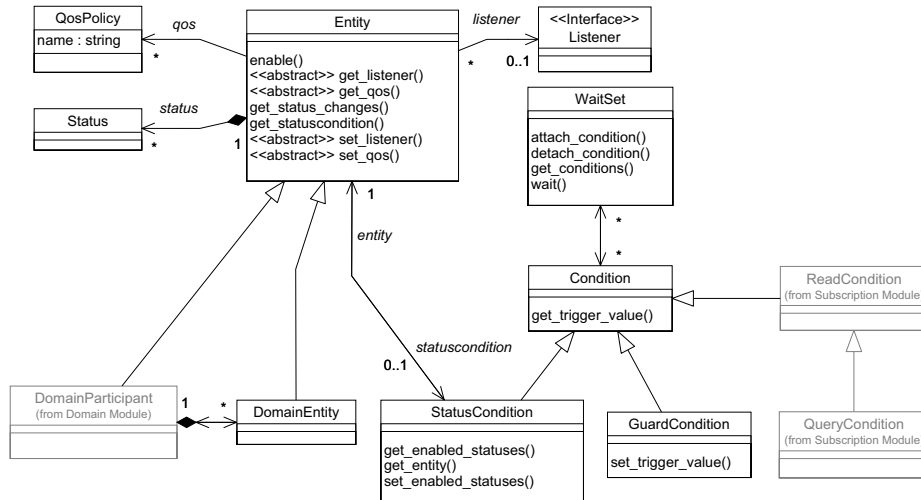


Figure 9 DCPS Infrastructure Module's Class Model

This module contains the following classes:

- DDS_Entity (abstract)
- DDS_DomainEntity (abstract)
- DDS_QosPolicy (abstract, struct)
- DDS_Listener (interface)
- DDS_Status (abstract, struct)
- DDS_WaitSet
- DDS_Condition
- DDS_GuardCondition
- DDS_StatusCondition

3.1.1 Class DDS_Entity (abstract)

This class is the abstract base class for all the DCPS objects. It acts as a generic class for DDS_Entity objects.

The interface description of this class is as follows:

```

/* interface DDS_Entity */
/* abstract operations (implemented in class
DDS_DomainParticipant,
 * DDS_Topic, DDS_Publisher, DDS_DataWriter, DDS_Subscriber and
 * DDS_DataReader)
 */
/*

```

```

* DDS_ReturnCode_t
*   DDS_Entity_set_qos
*       (DDS_Entity _this,
*        const DDS_EntityQos *qos);
*/
/*
* DDS_ReturnCode_t
*   DDS_Entity_get_qos
*       (DDS_Entity _this,
*        DDS_EntityQos *qos);
*/
/*
* DDS_ReturnCode_t
*   DDS_Entity_set_listener
*       (DDS_Entity _this,
*        const struct DDS_EntityListener *a_listener,
*        const DDS_StatusMask mask);
*/
/*
* struct DDS_EntityListener
*   DDS_Entity_get_listener
*       (DDS_Entity _this);
*/
/*
* implemented API operations
*/
    DDS_ReturnCode_t
        DDS_Entity_enable
            (DDS_Entity _this);
    DDS_StatusCondition
        DDS_Entity_get_statuscondition
            (DDS_Entity _this);
    DDS_StatusMask
        DDS_Entity_get_status_changes
            (DDS_Entity _this);
    DDS_InstanceHandle_t
        DDS_Entity_get_instance_handle
            (DDS_Entity _this);

```

The abstract operations are listed but not fully described because they are not implemented in this specific class. The full description of these operations is given in the subclasses, which contain the type specific implementation of these operations.

3.1.1.1 DDS_Entity_enable

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t

```

```
DDS_Entity_enable
(DDS_Entity _this);
```

Description

This operation enables the `DDS_Entity` on which it is being called when the `DDS_Entity` was created with the `DDS_EntityFactoryQosPolicy` set to `FALSE`.

Parameters

in `DDS_Entity _this` - the `DDS_Entity` object on which the operation is operated.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation enables the `DDS_Entity`. Created `DDS_Entity` objects can start in either an enabled or disabled state. This is controlled by the value of the `DDS_EntityFactoryQosPolicy` on the corresponding factory for the `DDS_Entity`. Enabled entities are immediately activated at creation time meaning all their immutable QoS settings can no longer be changed. Disabled Entities are not yet activated, so it is still possible to change their immutable QoS settings. However, once activated the immutable QoS settings can no longer be changed.

Creating disabled entities can make sense when the creator of the `DDS_Entity` does not yet know which QoS settings to apply, thus allowing another piece of code to set the QoS later on. This is for example the case in the DLRL, where the `ObjectHomes` create all underlying DCPS entities but do not know which QoS settings to apply. The user can then apply the required QoS settings afterwards.

The default setting of `DDS_EntityFactoryQosPolicy` is such that, by default, entities are created in an enabled state so that it is not necessary to explicitly call `DDS_<Entity>_enable` on newly created entities.

The `DDS_<Entity>_enable` operation produces the same results no matter how many times it is performed. Calling `DDS_<Entity>_enable` on an already enabled `DDS_Entity` returns `DDS_RETCODE_OK` and has no effect.

If a `DDS_Entity` has not yet been enabled, the only operations that can be invoked on it are: the ones to set, get or copy the `QosPolicy` settings (including the default `QosPolicy` settings on factories), the ones that set (or get) the listener, the ones that get the `DDS_StatusCondition`, the `DDS_Entity_get_status_changes`

operation (although the status of a disabled entity never changes), and the ‘factory’ operations that create, delete or lookup¹ other `DDS_Entities`. Other operations will return the error `DDS_RETCODE_NOT_ENABLED`.

Entities created from a factory that is disabled, are created disabled regardless of the setting of the `DDS_EntityFactoryQosPolicy`.

Calling `DDS_<Entity>_enable` on an `DDS_Entity` whose factory is not enabled will fail and return `DDS_RETCODE_PRECONDITION_NOT_MET`.

If the `DDS_EntityFactoryQosPolicy` has `autoenable_created_entities` set to `TRUE`, the `DDS_<Entity>_enable` operation on the factory will automatically enable all Entities created from the factory.

The `Listeners` associated with an `DDS_Entity` are not called until the `DDS_Entity` is enabled. `DDS_Conditions` associated with an `DDS_Entity` that is not enabled are "inactive", that is, have a `trigger_value` which is `FALSE`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the application enabled the `DDS_Entity` (or it was already enabled)
- `DDS_RETCODE_ERROR` - an internal error has occurred
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the factory of the `DDS_Entity` is not enabled

3.1.1.2 `DDS_Entity_get_instance_handle`

Synopsis

```
#include <dds_dcps.h>
DDS_InstanceHandle_t
    DDS_Entity_get_instance_handle
        (DDS_Entity _this);
```

Description

This operation returns the `instance_handle` of the built-in topic sample that represents the specified `DDS_Entity`.

1. This includes the `lookup_topicdescription`, but not the `find_topic`.

Parameters

in DDS_Entity _this - object on which the operation is operated.

Return Value

DDS_InstanceHandle_t - Result value is the *instance_handle* of the built-in topic sample that represents the state of this *DDS_Entity*.

Detailed Description

The relevant state of some *DDS_Entity* objects are distributed using built-in topics. Each built-in topic sample represents the state of a specific *DDS_Entity* and has a unique *instance_handle*. This operation returns the *instance_handle* of the built-in topic sample that represents the specified *DDS_Entity*.

Some *DDS_Entities* (*DDS_Publisher* and *DDS_Subscriber*) do not have a corresponding built-in topic sample, but they still have an *instance_handle* that uniquely identifies the *DDS_Entity*.

The *instance_handles* obtained this way can also be used to check whether a specific *DDS_Entity* is located in a specific *DDS_DomainParticipant*. (See Section 3.2.1.2, *DDS_DomainParticipant_contains_entity*, on page 142.)

3.1.1.3 DDS_Entity_get_listener (abstract)

This abstract operation is defined as a generic operation to access a *Listener*. Each subclass derived from this class, *DDS_DomainParticipant*, *DDS_Topic*, *DDS_Publisher*, *DDS_Subscriber*, *DDS_DataWriter* and *DDS_DataReader* will provide a class specific implementation of this abstract operation.

Synopsis

```
#include <dds_dcps.h>
struct DDS_EntityListener
    DDS_Entity_get_listener
        (DDS_Entity _this);
```

3.1.1.4 DDS_Entity_get_qos (abstract)

This abstract operation is defined as a generic operation to access a struct with the *QosPolicy* settings. Each subclass derived from this class, *DDS_DomainParticipant*, *DDS_Topic*, *DDS_Publisher*, *DDS_Subscriber*, *DDS_DataWriter* and *DDS_DataReader* will provide a class specific implementation of this abstract operation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Entity_get_qos
```

```
(DDS_Entity _this,
 DDS_EntityQos *qos);
```

3.1.1.5 DDS_Entity_get_status_changes

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
DDS_Entity_get_status_changes
(DDS_Entity _this);
```

Description

This operation returns a mask with the communication statuses in the `DDS_Entity` that are “triggered”.

Parameters

in `DDS_Entity _this` - object on which the operation is operated.

Return Value

`DDS_StatusMask` - Result is a bit-mask in which each bit shows which value has changed.

Detailed Description

This operation returns a mask with the communication statuses in the `DDS_Entity` that are “triggered”. That is the set of communication statuses whose value have changed since the last time the application called this operation. This operation shows whether a change has occurred even when the status seems unchanged because the status changed back to the original status.

When the `DDS_Entity` is first created or if the `DDS_Entity` is not enabled, all communication statuses are in the “un-triggered” state so the mask returned by the operation is empty.

The result value is a bit-mask in which each bit shows which value has changed. The relevant bits represent one of the following statuses:

- `DDS_INCONSISTENT_TOPIC_STATUS`
- `DDS_OFFERED_DEADLINE_MISSED_STATUS`
- `DDS_REQUESTED_DEADLINE_MISSED_STATUS`
- `DDS_OFFERED_INCOMPATIBLE_QOS_STATUS`
- `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS`
- `DDS_SAMPLE_LOST_STATUS`
- `DDS_SAMPLE_REJECTED_STATUS`
- `DDS_DATA_ON_READERS_STATUS`

- `DDS_DATA_AVAILABLE_STATUS`
- `DDS_LIVELINESS_LOST_STATUS`
- `DDS_LIVELINESS_CHANGED_STATUS`
- `DDS_PUBLICATION_MATCHED_STATUS`
- `DDS_SUBSCRIPTION_MATCHED_STATUS`

Each status bit is declared as a constant and can be used in an AND operation to check the status bit against the result of type `DDS_StatusMask`. Not all statuses are relevant to all `DDS_Entity` objects. See the respective `Listener` interfaces for each `DDS_Entity` for more information.

3.1.1.6 `DDS_Entity_get_statuscondition`

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
    DDS_Entity_get_statuscondition
        (DDS_Entity _this);
```

Description

This operation allows access to the `DDS_StatusCondition` associated with the `DDS_Entity`.

Parameters

in `DDS_Entity _this` - object on which the operation is operated.

Return Value

DDS_StatusCondition - Result value is the `DDS_StatusCondition` of the `DDS_Entity`.

Detailed Description

Each `DDS_Entity` has a `DDS_StatusCondition` associated with it. This operation allows access to the `DDS_StatusCondition` associated with the `DDS_Entity`. The returned condition can then be added to a `DDS_WaitSet` so that the application can wait for specific status changes that affect the `DDS_Entity`.

3.1.1.7 `DDS_Entity_set_listener` (abstract)

This abstract operation is defined as a generic operation to access a `Listener`. Each subclass derived from this class, `DDS_DomainParticipant`, `DDS_Topic`, `DDS_Publisher`, `DDS_Subscriber`, `DDS_DataWriter` and `DDS_DataReader` will provide a class specific implementation of this abstract operation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Entity_set_listener
        (DDS_Entity _this,
         const struct DDS_Listener *a_listener,
         const DDS_StatusMask mask);
```

3.1.1.8 DDS_Entity_set_qos (abstract)

This abstract operation is defined as a generic operation to modify a struct with the `QosPolicy` settings. Each subclass derived from this class, `DDS_DomainParticipant`, `DDS_Topic`, `DDS_Publisher`, `DDS_Subscriber`, `DDS_DataWriter` and `DDS_DataReader` will provide a class specific implementation of this abstract operation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Entity_set_qos
        (DDS_Entity _this,
         const DDS_EntityQos *qos);
```

3.1.2 Class DDS_DomainEntity (abstract)

This class is the abstract base class for the all entities except `DDS_DomainParticipant`. The main purpose is to express that `DDS_DomainParticipant` is a special kind of `DDS_Entity`, which acts as a container of all other `DDS_Entity` objects, but cannot contain another `DDS_DomainParticipant` within itself. Therefore, this class is not part of the IDL interface in the DCPS PSM description.

The class `DDS_DomainEntity` does not contain any operations.

3.1.3 Struct QosPolicy

Each `DDS_Entity` provides a `<DDS_Entity>Qos` structure that implements the basic mechanism for an application to specify Quality of Service attributes. This structure consists of `DDS_Entity` specific `QosPolicy` attributes. `QosPolicy` attributes are structured types where each type specifies the information that controls an `DDS_Entity` related (configurable) property of the Data Distribution Service.

All `QosPolicies` applicable to a `DDS_Entity` are aggregated in a corresponding `<DDS_Entity>Qos`, which is a compound structure that is set atomically so that it represents a coherent set of `QosPolicy` attributes.

Compound types are used whenever multiple attributes must be set coherently to define a consistent attribute for a `QosPolicy`.

See Appendix A, *Quality Of Service* for details of the `<DDS_Entity>Qos`, along with a complete list of individual `QosPolicy` settings and their meanings.

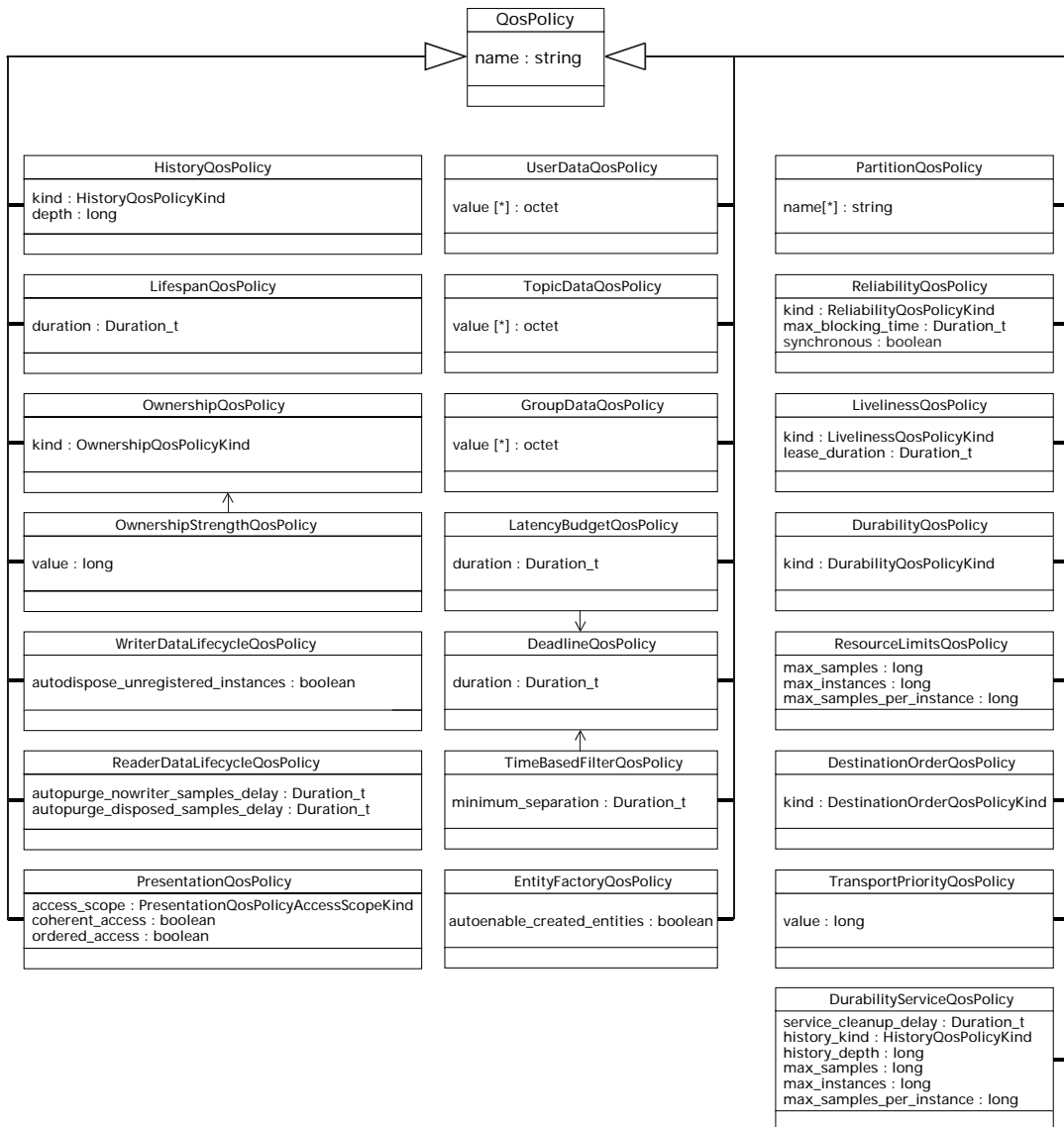


Figure 10 QosPolicy Settings

Requested/Offered

In several cases, for communications to occur properly (or efficiently), a `QosPolicy` on the requesting side must be compatible with a corresponding `QosPolicy` on the offering side. For example, if a `DDS_DataReader` requests to receive data reliably while the corresponding `DDS_DataWriter` defines a best-effort `QosPolicy`, communication will not happen as requested. This means that the specification for `QosPolicy` follows the requested/offered (RxO) pattern while trying to maintain the desirable decoupling of publication and subscription as much as possible. In this pattern:

- the requesting side can specify a “requested” attribute for a particular `QosPolicy`
- the offering side specifies an “offered” attribute for that `QosPolicy`.

The Data Distribution Service will then determine whether the attribute requested by the requesting side is compatible with what is offered by the offering side. Only when the two `QosPolicy` settings are compatible, communication is established. If the two `QosPolicy` settings are not compatible, the Data Distribution Service will not establish communication between the two `DDS_Entity` objects and notify this fact by means of the `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side and the `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side. The application can detect this fact by means of a `Listener` or `DDS_Condition`.

The interface description of these `QosPolicy`s are as follows:

```
/*
 * struct DDS_<DDS_Entity>Qos
 *     see appendix
 */
/*
 * struct DDS_<name>QosPolicy
 */
struct DDS_UserDataQosPolicy
{ DDS_sequence_octet value; };
struct DDS_TopicDataQosPolicy
{ DDS_sequence_octet value; };
struct DDS_GroupDataQosPolicy
{ DDS_sequence_octet value; };
struct DDS_TransportPriorityQosPolicy
{ DDS_long value; };
struct DDS_LifespanQosPolicy
{ DDS_Duration_t duration; };
enum DDS_DurabilityQosPolicyKind
{ DDS_VOLATILE_DURABILITY_QOS,
  DDS_TRANSIENT_LOCAL_DURABILITY_QOS,
  DDS_TRANSIENT_DURABILITY_QOS,
  DDS_PERSISTENT_DURABILITY_QOS };
struct DDS_DurabilityQosPolicy
{ DDS_DurabilityQosPolicyKind kind; };
```

```

enum DDS_PresentationQosPolicyAccessScopeKind
{
    DDS_INSTANCE_PRESENTATION_QOS,
    DDS_TOPIC_PRESENTATION_QOS,
    DDS_GROUP_PRESENTATION_QOS };
struct DDS_PresentationQosPolicy
{
    DDS_PresentationQosPolicyAccessScopeKind
        access_scope;
    DDS_boolean coherent_access;
    DDS_boolean ordered_access; };
struct DDS_DeadlineQosPolicy
{
    DDS_Duration_t period; };
struct DDS_LatencyBudgetQosPolicy
{
    DDS_Duration_t duration; };
enum DDS_OwnershipQosPolicyKind
{
    DDS_SHARED_OWNERSHIP_QOS,
    DDS_EXCLUSIVE_OWNERSHIP_QOS };
struct DDS_OwnershipQosPolicy
{
    DDS_OwnershipQosPolicyKind kind; };
struct DDS_OwnershipStrengthQosPolicy
{
    DDS_long value; };
enum DDS_LivelinessQosPolicyKind
{
    DDS_AUTOMATIC_LIVELINESS_QOS,
    DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS,
    DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS };
struct DDS_LivelinessQosPolicy
{
    DDS_LivelinessQosPolicyKind kind;
    DDS_Duration_t lease_duration; };
struct DDS_TimeBasedFilterQosPolicy
{
    DDS_Duration_t minimum_separation; };
struct DDS_PartitionQosPolicy
{
    DDS_StringSeq name; };
enum DDS_ReliabilityQosPolicyKind
{
    DDS_BEST_EFFORT_RELIABILITY_QOS,
    DDS_RELIABLE_RELIABILITY_QOS };
struct DDS_ReliabilityQosPolicy
{
    DDS_ReliabilityQosPolicyKind kind;
    DDS_Duration_t max_blocking_time;
    DDS_boolean synchronous; };
enum DDS_DestinationOrderQosPolicyKind
{
    DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS,
    DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS };
struct DDS_DestinationOrderQosPolicy
{
    DDS_DestinationOrderQosPolicyKind kind; };
enum DDS_HistoryQosPolicyKind
{
    DDS_KEEP_LAST_HISTORY_QOS,
    DDS_KEEP_ALL_HISTORY_QOS };
struct DDS_HistoryQosPolicy
{
    DDS_HistoryQosPolicyKind kind;
    DDS_long depth; };
struct DDS_ResourceLimitsQosPolicy

```

```

    { DDS_long max_samples;
      DDS_long max_instances;
      DDS_long max_samples_per_instance; };
struct DDS_EntityFactoryQosPolicy
    { DDS_boolean autoenable_created_entities; };
struct DDS_WriterDataLifecycleQosPolicy
    { DDS_boolean autodispose_unregistered_instances; };
struct DDS_ReaderDataLifecycleQosPolicy
    { DDS_Duration_t autopurge_nowriter_samples_delay;
      DDS_Duration_t autopurge_disposed_samples_delay; };
struct DurabilityServiceQosPolicy
    { DDS_Duration_t service_cleanup_delay;
      DDS_HistoryQosPolicyKind history_kind;
      DDS_long history_depth;
      DDS_long max_samples;
      DDS_long max_instances;
      DDS_long max_samples_per_instance; };
enum DDS_SchedulingClassQosPolicyKind
    { DDS_SCHEDULE_DEFAULT,
      DDS_SCHEDULE_TIMESHARING,
      DDS_SCHEDULE_REALTIME };
struct DDS_SchedulingClassQosPolicy
    { DDS_SchedulingClassQosPolicyKind kind; };
enum DDS_SchedulingPriorityQosPolicyKind
    { DDS_PRIORITY_RELATIVE,
      DDS_PRIORITY_ABSOLUTE };
struct DDS_SchedulingPriorityQosPolicy
    { DDS_SchedulingPriorityQosPolicyKind kind; };
struct DDS_SchedulingQosPolicy
    { DDS_SchedulingClassQosPolicy scheduling_class;
      DDS_SchedulingPriorityQosPolicy scheduling_priority_kind;
      DDS_long scheduling_priority; };
/*
 * implemented API operations
 * <no operations>
 */

```

Default attributes

The default attributes of each QosPolicy are listed in the next table:

Table 5 QosPolicy Default Attributes

QosPolicy	Attribute	Value
user_data	value.length	0
topic_data	value.length	0
group_data	value.length	0
transport_priority	value	0
lifespan	duration	DDS_DURATION_INFINITE

Table 5 QosPolicy Default Attributes (Continued)

QosPolicy	Attribute	Value
durability	kind	DDS_VOLATILE_DURABILITY_QOS
presentation	access_scope	DDS_INSTANCE_PRESENTATION_QOS
	coherent_access	FALSE
	ordered_access	FALSE
deadline	period	DDS_DURATION_INFINITE
latency_budget	duration	0
ownership_strength	value	0
ownership	kind	DDS_SHARED_OWNERSHIP_QOS
liveliness	kind	DDS_AUTOMATIC_LIVELINESS_QOS
	lease_duration	DDS_DURATION_INFINITE
time_based_filter	minimum_separation	0
partition	name.length	0
reliability	kind	DDS_BEST_EFFORT_RELIABILITY_QOS
	max_blocking_time	100 ms
	synchronous	FALSE
destination_order	kind	DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
history	kind	DDS_KEEP_LAST_HISTORY_QOS
	depth	1
resource_limits	max_samples	DDS_LENGTH_UNLIMITED
	max_instances	DDS_LENGTH_UNLIMITED
	max_samples_per_instance	DDS_LENGTH_UNLIMITED
entity_factory	autoenable_created_entities	TRUE
writer_data_lifecycle	autodispose_unregistered_instances	TRUE
reader_data_lifecycle	autopurge_nowriter_samples_delay	DDS_DURATION_INFINITE
	autopurge_disposed_samples_delay	DDS_DURATION_INFINITE

Table 5 QosPolicy Default Attributes (Continued)

QosPolicy	Attribute	Value
durability_service	history_kind	KEEP_LAST
	history_depth	1
	max_samples	LENGTH_UNLIMITED
	max_instances	LENGTH_UNLIMITED
	max_samples_per_instance	LENGTH_UNLIMITED
	service_cleanup_delay	0
watchdog_scheduling, listener_scheduling	scheduling_class.kind	DDS_SCHEDULE_DEFAULT
	scheduling_priority_kind. kind	DDS_PRIORITY_RELATIVE
	scheduling_priority	0

RxO

The QosPolicy settings that need to be set in a compatible manner between the publisher and subscriber ends are indicated by the setting of the “RxO” (Requested/Offered) property. The “RxO” property of each QosPolicy is listed in Table 6 on page 62

- A “RxO” setting of “Yes” indicates that the QosPolicy can be set at both ends (publishing and subscribing) and the attributes must be set in a compatible manner. In this case the compatible attributes are explicitly defined
- A “RxO” setting of “No” indicates that the QosPolicy can be set at both ends (publishing and subscribing) but the two settings are independent. That is, all combinations of attributes are compatible
- A “RxO” setting of “Not applicable” indicates that the QosPolicy can only be specified at either the publishing or the subscribing end, but not at both ends. So compatibility does not apply.

Changeable

The “changeable” property determines whether the QosPolicy can be changed after the DDS_Entity is enabled. In other words, a QosPolicy with “changeable” setting of “No” is considered “immutable” and can only be specified either at DDS_Entity creation time or prior to calling the DDS_Entity_enable operation on the DDS_Entity.

When the application tries to change a QosPolicy with “changeable” setting of “No”, the Data Distribution Service will notify this by returning a DDS_RETCODE_IMMUTABLE_POLICY.

The basic way to modify or set the `<DDS_Entity>Qos` is by using a `DDS_<Entity>_get_qos` and `DDS_<Entity>_set_qos` operation to get all `QosPolicy` settings from this `DDS_Entity` (that is the `<DDS_Entity>Qos`), modify several specific `QosPolicy` settings and put them back using an user operation to set all `QosPolicy` settings on this `DDS_Entity` (that is the `<DDS_Entity>Qos`). An example of these operations for the `DDS_DataWriter` are `DDS_DataWriter_get_qos` and `DDS_DataWriter_set_qos`, which take the `DataWriterQos` as a parameter.

The “RxO” setting and the “changeable” setting of each `QosPolicy` are listed in the next table:

Table 6 QosPolicy Basics

QosPolicy	Concerns DDS_Entity	RxO	Changeable After Enabling
user_data	DDS_DomainParticipant DDS_DataReader DDS_DataWriter	No	Yes
topic_data	DDS_Topic	No	Yes
group_data	DDS_Publisher DDS_Subscriber	No	Yes
transport_priority	DDS_Topic DDS_DataWriter	Not applicable	Yes
lifespan	DDS_Topic DDS_DataWriter	Not applicable	Yes
durability	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	No
presentation	DDS_Publisher DDS_Subscriber	Yes	No
deadline	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	Yes
latency_budget	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	Yes
ownership	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	No
ownership_strength	DDS_DataWriter	Not applicable	Yes

Table 6 QosPolicy Basics (Continued)

QosPolicy	Concerns DDS_Entity	RxO	Changeable After Enabling
liveliness	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	No
time_based_filter	DDS_DataReader	Not applicable	Yes
partition	DDS_Publisher DDS_Subscriber	No	Yes
reliability	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	No
destination_order	DDS_Topic DDS_DataReader DDS_DataWriter	Yes	No
history	DDS_Topic DDS_DataReader DDS_DataWriter	No	No
resource_limits	DDS_Topic DDS_DataReader DDS_DataWriter	No	No
entity_factory	DDS_DomainParticipantFactory DDS_DomainParticipant DDS_Publisher DDS_Subscriber	No	Yes
writer_data_lifecycle	DDS_DataWriter	Not applicable	Yes
reader_data_lifecycle	DDS_DataReader	Not applicable	Yes
durability_service	DDS_Topic	No	No
scheduling	DDS_DomainParticipant	Not applicable	No

The next paragraphs describe the usage of each QosPolicy struct.

3.1.3.1 DDS_DeadlineQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_DeadlineQosPolicy
{ DDS_Duration_t period; };
```

Description

This `QosPolicy` defines the period within which a new sample is expected by the `DataReader` or to be written by the `DataWriter`.

Attributes

`DDS_Duration_t period` - specifies the period within which a new sample is expected or to be written.

Detailed Description

This `QosPolicy` will set the period within which a `DDS_DataReader` expects a new sample or, in case of a `DDS_DataWriter`, the period in which it expects applications to write the sample. The default value of the period is `DDS_DURATION_INFINITE`, indicating that there is no deadline. The `QosPolicy` may be used to monitor the real-time behaviour, a `DDS_Listener` or a `DDS_StatusCondition` may be used to catch the event that is generated when a deadline is missed.

`DDS_DeadlineQosPolicy` is instance oriented (i.e. the period is monitored for each individual instance).

The exact consequences of a missed deadline depend on the `DDS_Entity` in which it occurred, and the `DDS_OwnershipQosPolicy` value of that `DDS_Entity`:

- In case a `DDS_DataWriter` misses an instance deadline (regardless of its `DDS_OwnershipQosPolicy` setting), an `offered_deadline_missed_status` is raised, which can be detected by either a `DDS_Listener` or a `DDS_StatusCondition`. There are no further consequences.
- In case a `DDS_DataReader` misses an instance deadline, a `requested_deadline_missed_status` is raised, which can be detected by either a `DDS_Listener` or a `DDS_StatusCondition`. In case the `DDS_OwnershipQosPolicy` is set to `SHARED`, there are no further consequences. In case the `DDS_OwnershipQosPolicy` is set to `EXCLUSIVE`, the ownership of that instance on that particular `DDS_DataReader` is transferred to the next available highest strength `DDS_DataWriter`, but this will have no impact on the `instance_state` whatsoever. So even when a deadline is missed for an instance that has no other (lower-strength) `DDS_DataWriters` to transfer ownership to, the `instance_state` remains unchanged. See also Section 3.1.3.11, *DDS_OwnershipQosPolicy*.

This `QosPolicy` is applicable to a `DDS_DataReader`, a `DDS_DataWriter` and a `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` may be changed by using the `DDS_<DDS_Entity>_set_qos` operation.

Requested/Offered

In case the Requested/Offered `QoSPolicy` are incompatible, the notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side and `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side is raised.

Table 7 DDS_DeadlineQoSPolicy

Period	Compatibility
offered period < requested period	compatible
offered period = requested period	compatible
offered period > requested period	Incompatible

Whether communication is established, is controlled by the Data Distribution Service, depending on the Requested/Offered `QoSPolicy` of the `DDS_DataWriter` and `DDS_DataReader`. In other words, the communication between any `DDS_DataWriter` and `DDS_DataReader` depends on what is expected by the `DDS_DataReader`. As a consequence, a `DDS_DataWriter` that has an incompatible `QoS` with respect to what a `DDS_DataReader` specifies is not allowed to send its data to that specific `DDS_DataReader`. A `DDS_DataReader` that has an incompatible `QoS` with respect to what a `DDS_DataWriter` specifies does not get any data from that particular `DDS_DataWriter`.

Changing an existing deadline period using the `set_qos` operation on either the `DDS_DataWriter` or `DDS_DataReader` may have consequences for the connectivity between readers and writers, depending on their `RxO` values. (See also in Section 3.1.3, *Struct QoSPolicy*, the paragraph entitled *Requested/Offered*.) Consider a writer with deadline period P_w and a reader with deadline period P_r , where $P_w \leq P_r$. In this case a connection between that reader and that writer is established. Now suppose P_w is changed so that $P_w > P_r$, then the existing connection between reader and writer will be lost, and the reader will behave as if the writer unregistered all its instances, transferring the ownership of these instances when appropriate. See also Section 3.1.3.11, *DDS_OwnershipQoSPolicy*.

DDS_TopicQoS

This `QoSPolicy` can be set on a `DDS_Topic`. The `DDS_DataWriter` and/or `DDS_DataReader` can copy this `qos` by using the operations `DDS_<DDS_Entity>_copy_from_topic_qos` and then `DDS_<DDS_Entity>_set_qos`. That way the application can relatively easily ensure the `QoSPolicy` for the `DDS_Topic`, `DDS_DataReader` and `DDS_DataWriter` are consistent.

3.1.3.2 DDS_DestinationOrderQosPolicy

Synopsis

```
#include <dds_dcps.h>
enum DDS_DestinationOrderQosPolicyKind
{ DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS,
  DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS };
struct DDS_DestinationOrderQosPolicy
{ DDS_DestinationOrderQosPolicyKind kind; };
```

Description

This QosPolicy controls the order in which the DDS_DataReader stores the data.

Attributes

DDS_DestinationOrderQosPolicyKind kind - controls the order in which the DDS_DataReader stores the data.

Detailed Description

This QosPolicy controls the order in which the DDS_DataReader stores the data. The order of storage is controlled by the timestamp. However a choice can be made to use the timestamp of the DDS_DataReader (time of reception) or the timestamp of the DDS_DataWriter (source timestamp).

This QosPolicy is applicable to a DDS_DataWriter, DDS_DataReader and a DDS_Topic. After enabling of the concerning DDS_Entity, this QosPolicy cannot be changed any more.

Attribute

The QosPolicy is controlled by the attribute kind which may be:

- DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
- DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS

When set to DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS, the order is based on the timestamp, at the moment the sample was received by the DDS_DataReader.

When set to DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS, the order is based on the timestamp, which was set by the DDS_DataWriter. This means that the system needs some time synchronisation.

Requested/Offered

In case the Requested/Offered QoSPolicy are incompatible, the notification DDS_OFFERED_INCOMPATIBLE_QOS status on the offering side and DDS_REQUESTED_INCOMPATIBLE_QOS status on the requesting side is raised.

Table 8 Requested/Offered DDS_DestinationOrderQoSPolicy

Requested Offered	BY_RECEPTION_TIMESTAMP	BY_SOURCE_TIMESTAMP
BY_RECEPTION_TIMESTAMP	compatible	Incompatible
BY_SOURCE_TIMESTAMP	compatible	compatible

Whether communication is established, is controlled by the Data Distribution Service, depending on the Requested/Offered QoSPolicy of the DDS_DataWriter and DDS_DataReader. In other words, the communication between any DDS_DataWriter and DDS_DataReader depends on what is expected by the DDS_DataReader. As a consequence, a DDS_DataWriter that has an incompatible QoS with respect to what a DDS_DataReader specified, is not allowed to send its data to that specific DDS_DataReader. A DDS_DataReader that has an incompatible QoS with respect to what a DDS_DataWriter specified, does not get any data from that particular DDS_DataWriter.

DDS_TopicQos

This QoSPolicy can be set on a DDS_Topic. The DDS_DataWriter and/or DDS_DataReader can copy this qos by using the operations DDS_<DDS_Entity>_copy_from_topic_qos and then DDS_<DDS_Entity>_set_qos. That way the application can relatively easily ensure the QoSPolicy for the DDS_Topic, DDS_DataReader and DDS_DataWriter are consistent.

3.1.3.3 DDS_DurabilityQoSPolicy**Synopsis**

```
#include <dds_dcps.h>
enum DDS_DurabilityQoSPolicyKind
{
    DDS_VOLATILE_DURABILITY_QOS,
    DDS_TRANSIENT_LOCAL_DURABILITY_QOS,
    DDS_TRANSIENT_DURABILITY_QOS,
    DDS_PERSISTENT_DURABILITY_QOS };
struct DDS_DurabilityQoSPolicy
{
    DDS_DurabilityQoSPolicyKind kind;
};
```

Description

This QoSPolicy controls whether the data should be stored for late joining readers.

Attributes

DDS_DurabilityQosPolicyKind kind - specifies the type of durability from `DDS_VOLATILE_DURABILITY_QOS` (short life) to `DDS_PERSISTENT_DURABILITY_QOS` (long life).

Detailed Description

The decoupling between `DDS_DataReader` and `DDS_DataWriter` offered by the Data Distribution Service allows an application to write data even if there are no current readers on the network. Moreover, a `DDS_DataReader` that joins the network after some data has been written could potentially be interested in accessing the most current values of the data as well as some history. This `QosPolicy` controls whether the Data Distribution Service will actually make data available to late-joining `DDS_DataReaders`.

This `QosPolicy` is applicable to a `DDS_DataReader`, `DDS_DataWriter` and `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` cannot be changed any more.

Attributes

The `QosPolicy` is controlled by the attribute `kind` which may be:

- *DDS_VOLATILE_DURABILITY_QOS* - the samples are not available to late-joining `DDS_DataReaders`. In other words, only `DDS_DataReaders`, which were present at the time of the writing and have subscribed to this `DDS_Topic`, will receive the sample. When a `DDS_DataReader` subscribes afterwards (late-joining), it will only be able to read the next written sample. This setting is typically used for data, which is updated quickly;
- *DDS_TRANSIENT_LOCAL_DURABILITY_QOS* - currently behaves identically to the `TRANSIENT_DURABILITY_QOS`, except for its `RxO` properties. The desired behaviour of `TRANSIENT_LOCAL_DURABILITY_QOS` can be achieved from the `TRANSIENT_DURABILITY_QOS` with the default (`TRUE`) setting of the `autodispose_unregistered_instances` flag on the `DataWriter` and the `service_cleanup_delay` set to 0 on the durability service. This is because for `TRANSIENT_LOCAL`, the data should only remain available for late-joining readers during the lifetime of its source writer, so it is not required to survive after its source writer has been deleted. Since the deletion of a writer implicitly unregisters all its instances, an `autodispose_unregistered_instances` value of `TRUE` will also dispose the affected data from the durability store, and thus prevent it from remaining available to late-joining readers.
- *DDS_TRANSIENT_DURABILITY_QOS* - some samples are available to late-joining `DDS_DataReaders` (stored in memory). This means that the late-joining `DDS_DataReaders` are able to read these previously written samples. The

DDS_DataReader does not necessarily have to exist at the time of writing. Not all samples are stored (depending on QosPolicy History and QosPolicy resource_limits). The storage does not depend on the DDS_DataWriter and will outlive the DDS_DataWriter. This may be used to implement reallocation of applications because the data is saved in the Data Distribution Service (not in the DDS_DataWriter). This setting is typically used for state related information of an application. In this case also the DurabilityServiceQosPolicy settings are relevant for the behaviour of the Data Distribution Service;

- *DDS_PERSISTENT_DURABILITY_QOS* - the data is stored in permanent storage (e.g. hard disk). This means that the samples are also available after a system restart. The samples not only outlives the DDS_DataWriters, but even the Data Distribution Service and the system. This setting is typically used for attributes and settings for an application or the system. In this case also the DurabilityServiceQosPolicy settings are relevant for the behaviour of the Data Distribution Service.

Requested/Offered

In case the Requested/Offered QosPolicy are incompatible, the notification DDS_OFFERED_INCOMPATIBLE_QOS status on the offering side and DDS_REQUESTED_INCOMPATIBLE_QOS status on the requesting side is raised.

Table 9 Requested/Offered DDS_DurabilityQosPolicy

Requested Offered	VOLATILE	TRANSIENT_ LOCAL	TRANSIENT	PERSISTENT
VOLATILE	compatible	INcompatible	INcompatible	INcompatible
TRANSIENT_LOCAL	compatible	compatible	INcompatible	INcompatible
TRANSIENT	compatible	compatible	compatible	INcompatible
PERSISTENT	compatible	compatible	compatible	compatible

This means that the Request/Offering mechanism is applicable between:

- the DDS_DataWriter and the DDS_DataReader: if the QosPolicy settings between DDS_DataWriter and DDS_DataReader are inconsistent, no communication between them is established. In addition the DDS_DataWriter will be informed via a DDS_REQUESTED_INCOMPATIBLE_QOS status change and the DDS_DataReader will be informed via an DDS_OFFERED_INCOMPATIBLE_QOS status change
- the DDS_DataWriter and the Data Distribution Service (as a built-in DDS_DataReader): if the QosPolicy settings between DDS_DataWriter and the Data Distribution Service are inconsistent, no communication between them is established. In that case data published by the DDS_DataWriter will not be

maintained by the service and as a consequence will not be available for late joining DDS_DataReaders. The QosPolicy of the Data Distribution Service in the role of DDS_DataReader is specified by the DDS_Topic QosPolicy

- the Data Distribution Service (as a built-in DDS_DataWriter) and the DDS_DataReader: if the QosPolicy settings between the Data Distribution Service and the DDS_DataReader are inconsistent, no communication between them is established. In that case the Data Distribution Service will not publish historical data to late joining DDS_DataReaders. The QosPolicy of the Data Distribution Service in the role of DDS_DataWriter is specified by the DDS_Topic QosPolicy.

Cleanup

The DDS_DurabilityQosPolicy kind setting DDS_TRANSIENT_LOCAL_DURABILITY_QOS, DDS_TRANSIENT_DURABILITY_QOS and DDS_PERSISTENT_DURABILITY_QOS determine that the DDS_DurabilityServiceQosPolicy applies for the DDS_Topic. It controls amongst others at which time the durability service is allowed to remove all information regarding a data-instance. Information on a data-instance is maintained until the following conditions are met:

- the instance has been explicitly disposed of (`instance_state = DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`),
- *and* the system detects that there are no more “live” DDS_DataWriter objects writing the instance, that is, all DDS_DataWriter either `unregister_instance` the instance (call `DDS_DataWriter_unregister_instance` operation) or lose their liveliness,
- *and* a time interval longer than `service_cleanup_delay` has elapsed since the moment the Data Distribution Service detected that the previous two conditions were met.

The use of the DDS_DurabilityServiceQosPolicy attribute `service_cleanup_delay` is apparent in the situation where an application disposes of an instance and it crashes before having a chance to complete additional tasks related to the disposition. Upon re-start the application may ask for initial data to regain its state and the delay introduced by the `service_cleanup_delay` allows the re-started application to receive the information on the disposed of instance and complete the interrupted tasks.

DDS_TopicQos

This `QosPolicy` can be set on a `DDS_Topic`. The `DDS_DataWriter` and/or `DDS_DataReader` can copy this `qos` by using the operations `DDS_<DDS_Entity>_copy_from_topic_qos` and then `DDS_<DDS_Entity>_set_qos`. That way the application can relatively easily ensure the `QosPolicy` for the `DDS_Topic`, `DDS_DataReader` and `DDS_DataWriter` are consistent.

3.1.3.4 DDS_DurabilityServiceQosPolicy**Scope**

DDS

Synopsis

```
#include <dds_dcps.h>
struct DDS_DurabilityServiceQosPolicy
{
    DDS_Duration_t service_cleanup_delay;
    DDS_HistoryQosPolicyKind history_kind;
    DDS_long history_depth;
    DDS_long max_samples;
    DDS_long max_instances;
    DDS_long max_samples_per_instance; };
```

Description

This `QosPolicy` controls the behaviour of the durability service regarding transient and persistent data.

Attributes

DDS_Duration_t service_cleanup_delay - specifies how long the durability service must wait before it is allowed to remove the information on the transient or persistent topic data-instances as a result of incoming dispose messages.

DDS_HistoryQosPolicyKind history_kind - specifies the type of history, which may be `DDS_KEEP_LAST_HISTORY_QOS`, or `DDS_KEEP_ALL_HISTORY_QOS` the durability service must apply for the transient or persistent topic data-instances.

DDS_long history_depth - specifies the number of samples of each instance of data (identified by its key) that is managed by the durability service for the transient or persistent topic data-instances. If *history_kind* is `KEEP_LAST_HISTORY_QOS`, *history_depth* must be smaller than or equal to *max_samples_per_instance* for this `QosPolicy` to be consistent.

DDS_long max_samples - specifies the maximum number of data samples for all instances the durability service will manage for the transient or persistent topic data-instances.

DDS_long max_instances - specifies the maximum number of instances the durability service will manage for the transient or persistent topic data-instances.

DDS_long max_samples_per_instance - specifies the maximum number of samples of any single instance the durability service will manage for the transient or persistent topic data-instances. If *history_kind* is *DDS_KEEP_LAST_HISTORY_QOS*, *max_samples_per_instance* must be greater than or equal to *history_depth* for this *QosPolicy* to be consistent.

Detailed Description

This *QosPolicy* controls the behaviour of the durability service regarding transient and persistent data. It controls for the transient or persistent topic; the time at which information regarding the topic may be discarded, the history policy it must set and the resource limits it must apply.

Cleanup

The setting of the *DDS_DurabilityServiceQosPolicy* only applies when *kind* of the *DDS_DurabilityQosPolicy* is either *DDS_TRANSIENT_DURABILITY_QOS* or *DDS_PERSISTENT_DURABILITY_QOS*. The *service_cleanup_delay* setting controls at which time the durability service is allowed to remove all information regarding a data-instance. Information on a data-instance is maintained until the following conditions are met:

- the instance has been explicitly disposed of (*instance_state* = *DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE*)
- **and** the system detects that there are no more “live” *DataWriter* objects writing the instance, that is, all *DataWriter* either *unregister_instance* the instance (call *unregister_instance* operation) or lose their liveliness
- **and** a time interval longer than *service_cleanup_delay* has elapsed since the moment the Data Distribution Service detected that the previous two conditions were met

The use of the attribute *service_cleanup_delay* is apparent in the situation where an application disposes an instance and it crashes before having a chance to complete additional tasks related to the disposal of the instance. Upon re-start the application may ask for initial data to regain its state and the delay introduced by the *service_cleanup_delay* allows the re-started application to receive the information of the disposed instance and complete the interrupted tasks.

History

The attributes `history_kind` and `history_depth` apply to the history settings of the durability service's internal `DDS_DataWriter` and `DDS_DataReader` managing the topic. The `DDS_HistoryQosPolicy` behaviour, as described in Section 3.1.3.7, *DDS_HistoryQosPolicy*, applies to these attributes.

Resource Limits

The attributes `max_samples`, `max_instances` and `max_samples_per_instance` apply to the resource limits of the durability service's internal `DDS_DataWriter` and `DDS_DataReader` managing the topic. The `DDS_ResourceLimitsQosPolicy` behaviour, as described in paragraph 3.1.3.17 (*DDS_ResourceLimitsQosPolicy*) applies to these attributes.

TopicQos

This `QosPolicy` can be set on a `DDS_Topic` only. After enabling of the concerning `DDS_Topic`, this `QosPolicy` can not be changed any more.

3.1.3.5 DDS_EntityFactoryQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_EntityFactoryQosPolicy
{ DDS_boolean autoenable_created_entities; };
```

Description

This `QosPolicy` controls the behaviour of the `Entity` as a factory for other entities.

Attributes

DDS_boolean autoenable_created_entities - specifies whether the entity acting as a factory automatically enables the instances it creates. If `autoenable_created_entities` is `TRUE` the factory will automatically enable each created `Entity`, otherwise it will not.

Detailed Description

This `QosPolicy` controls the behaviour of the `Entity` as a factory for other entities. It concerns only `DDS_DomainParticipantFactory` (as factory for `DDS_DomainParticipant`), `DDS_DomainParticipant` (as factory for `DDS_Publisher`, `DDS_Subscriber`, and `DDS_Topic`), `DDS_Publisher` (as factory for `DDS_DataWriter`), and `DDS_Subscriber` (as factory for `DDS_DataReader`).

This policy is mutable. A change in the policy affects only the entities created after the change; not the previously created entities.

The setting of `autoenable_created_entities` to `TRUE` indicates that the factory `create_<entity>` operation will automatically invoke the `enable` operation each time a new `DDS_Entity` is created. Therefore, the `DDS_Entity` returned by `create_<entity>` will already be enabled. A setting of `FALSE` indicates that the `DDS_Entity` will not be automatically enabled: the application will need to enable it explicitly by means of the `enable` operation. See Section 3.1.1.1, *DDS_Entity_enable* for a detailed description about the differences between enabled and disabled entities.

The default setting of `autoenable_created_entities` is `TRUE` meaning that by default it is not necessary to explicitly call `enable` on newly created entities.

3.1.3.6 DDS_GroupDataQosPolicy

Synopsis

```
#include <dds_dcps.h>
    struct DDS_GroupDataQosPolicy
    { DDS_sequence_octet value; };
```

Description

This `QosPolicy` allows the application to attach additional information to a `DDS_Publisher` or `DDS_Subscriber` `DDS_Entity`. This information is distributed with the `DDS_BuiltinTopics`.

Attributes

DDS_sequence_octet value - a sequence of octets that holds the application group data. By default, the sequence has length 0.

Detailed Description

This `QosPolicy` allows the application to attach additional information to a `DDS_Publisher` or `DDS_Subscriber` `DDS_Entity`. This information is distributed with the `DDS_BuiltinTopic`. An application that discovers a new `DDS_Entity` of the listed kind, can use this information to add additional functionality. The `DDS_GroupDataQosPolicy` is changeable and updates of the `DDS_BuiltinTopic` instance must be expected. Note that the Data Distribution Service is not aware of the real structure of the group data (the Data Distribution System handles it as an opaque type) and that the application is responsible for correct mapping on structural types for the specific platform.

3.1.3.7 DDS_HistoryQosPolicy

Synopsis

```
#include <dds_dcps.h>
enum DDS_HistoryQosPolicyKind
{
    DDS_KEEP_LAST_HISTORY_QOS,
    DDS_KEEP_ALL_HISTORY_QOS };
struct DDS_HistoryQosPolicy
{
    DDS_HistoryQosPolicyKind kind;
    DDS_long depth; };
```

Description

This `QosPolicy` controls which samples will be stored when the value of an instance changes (one or more times) before it is finally communicated.

Attributes

DDS_HistoryQosPolicyKind kind - specifies the type of history, which may be `DDS_KEEP_LAST_HISTORY_QOS` or `DDS_KEEP_ALL_HISTORY_QOS`.

DDS_long depth - specifies the number of samples of each instance of data (identified by its key) managed by this `DDS_Entity`.

Detailed Description

This `QosPolicy` controls whether the Data Distribution Service should deliver only the most recent sample, attempt to deliver all samples, or do something in between. In other words, how the `DDS_DataWriter` or `DDS_DataReader` should store samples. Normally, only the most recent sample is available but some history can be stored.

DDS_DataWriter

On the publishing side this `QosPolicy` controls the samples that should be maintained by the `DDS_DataWriter` on behalf of existing `DDS_DataReader` objects. The behaviour with respect to a `DDS_DataReader` objects discovered after a sample is written is controlled by the `DDS_DurabilityQosPolicy`.

DDS_DataReader

On the subscribing side it controls the samples that should be maintained until the application “takes” them from the Data Distribution Service.

This `QosPolicy` is applicable to a `DDS_DataReader`, `DDS_DataWriter` and `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` cannot be changed any more.

Attributes

The `QosPolicy` is controlled by the attribute kind which can be:

- `DDS_KEEP_LAST_HISTORY_QOS` - the Data Distribution Service will only attempt to keep the latest values of the instance and discard the older ones. The attribute “depth” determines how many samples in history will be stored. In other words, only the most recent samples in history are stored. On the publishing side, the Data Distribution Service will only keep the most recent “depth” samples of each instance of data (identified by its key) managed by the `DDS_DataWriter`. On the subscribing side, the `DDS_DataReader` will only keep the most recent “depth” samples received for each instance (identified by its key) until the application “takes” them via the `DDS_DataReader_take` operation. `DDS_KEEP_LAST_HISTORY_QOS` is the default kind. The default value of depth is 1, indicating that only the most recent value should be delivered. If a depth other than 1 is specified, it should be compatible with the settings of the `DDS_ResourceLimitsQosPolicy` `max_samples_per_instance`. For these two `QosPolicy` settings to be compatible, they must verify that `depth <= max_samples_per_instance`, otherwise a `DDS_RETCODE_INCONSISTENT_POLICY` is generated on relevant operations;
- `DDS_KEEP_ALL_HISTORY_QOS` - all samples are stored, provided, the resources are available. On the publishing side, the Data Distribution Service will attempt to keep all samples (representing each value written) of each instance of data (identified by its key) managed by the `DDS_DataWriter` until they can be delivered to all subscribers. On the subscribing side, the Data Distribution Service will attempt to keep all samples of each instance of data (identified by its key) managed by the `DDS_DataReader`. These samples are kept until the application “takes” them from the Data Distribution Service via the `DDS_DataReader_take` operation. The setting of depth has no effect. Its implied value is `DDS_LENGTH_UNLIMITED`. The resources that the Data Distribution Service can use to keep this history are limited by the settings of the `DDS_ResourceLimitsQosPolicy`. If the limit is reached, the behaviour of the Data Distribution Service will depend on the `DDS_ReliabilityQosPolicy`. If the `DDS_ReliabilityQosPolicy` is `DDS_BEST_EFFORT_RELIABILITY_QOS`, the old values are discarded. If `DDS_ReliabilityQosPolicy` is `DDS_RELIABLE_RELIABILITY_QOS`, the Data Distribution Service will block the `DDS_DataWriter` until it can deliver the necessary old values to all subscribers.

On the subscribing side it controls the samples that should be maintained until the application “takes” them from the Data Distribution Service. On the publishing side this `QosPolicy` controls the samples that should be maintained by the `DDS_DataWriter` on behalf of `DDS_DataReader` objects. The behaviour with respect to a `DDS_DataReader` objects discovered after a sample is written is

controlled by the `DDS_DurabilityQosPolicy`. In more detail, this `QosPolicy` specifies the behaviour of the Data Distribution Service in case the value of a sample changes (one or more times) before it can be successfully communicated to one or more `DDS_Subscribers`.

Requested/Offered

The setting of the `QosPolicy` offered is independent of the one requested, in other words they are never considered incompatible. The communication will not be rejected on account of this `QosPolicy`. The notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side or `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side will not be raised.

DDS TopicQos

This `QosPolicy` can be set on a `DDS_Topic`. The `DDS_DataWriter` and/or `DDS_DataReader` can copy this `qos` by using the operations `DDS_<DDS_Entity>_copy_from_topic_qos` and then `DDS_<DDS_Entity>_set_qos`. That way the application can relatively easily ensure the `QosPolicy` for the `DDS_Topic`, `DDS_DataReader` and `DDS_DataWriter` are consistent.

3.1.3.8 DDS_LatencyBudgetQosPolicy

Synopsis

```
#include <dds_dcps.h>
    struct DDS_LatencyBudgetQosPolicy
    { DDS_Duration_t duration; };
```

Description

Specifies the maximum acceptable additional delay to the typical transport delay from the time the data is written until the data is delivered at the `DDS_DataReader` and the application is notified of this fact.

Attributes

DDS_Duration_t duration - specifies the maximum acceptable additional delay from the time the data is written until the data is delivered.

Detailed Description

This `QosPolicy` specifies the maximum acceptable additional delay to the typical transport delay from the time the data is written until the data is delivered at the `DDS_DataReader` and the application is notified of this fact. This `QosPolicy` provides a means for the application to indicate to the Data Distribution Service the

“urgency” of the data-communication. By having a non-zero duration the Data Distribution Service can optimise its internal operation. The default value of the duration is zero, indicating that the delay should be minimized.

This `QosPolicy` is applicable to a `DDS_DataReader`, `DDS_DataWriter` and `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` may be changed by using the `DDS_<DDS_Entity>_set_qos` operation.

Requested/Offered

This `QosPolicy` is considered a hint to the Data Distribution Service, which will automatically adapt its behaviour to meet the requirements of the shortest delay if possible. In case the Requested/Offered `QosPolicy` are incompatible, the notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side and `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side is raised.

Table 10 DDS_LatencyBudgetQosPolicy

Duration	Compatibility
offered duration < requested duration	compatible
offered duration = requested duration	compatible
offered duration > requested duration	Incompatible

Note that even when the offered duration is considered compatible to the requested duration, this duration is not enforced in any way: there will be no notification on any violations of the requested duration.

Changing an existing latency budget using the `set_qos` operation on either the `DDS_DataWriter` or `DDS_DataReader` may have consequences for the connectivity between readers and writers, depending on their `RxO` values. (See also in Section 3.1.3, *Struct QosPolicy* the paragraph entitled *Requested/Offered*.) Consider a writer with budget B_w and a reader with budget B_r , where $B_w \leq B_r$. In this case a connection between that reader and that writer is established. Now suppose B_w is changed so that $B_w > B_r$, then the existing connection between reader and writer will be lost, and the reader will behave as if the writer unregistered all its instances, transferring the ownership of these instances when appropriate. See also Section 3.1.3.11, *DDS_OwnershipQosPolicy*.

DDS_TopicQos

This `QosPolicy` can be set on a `DDS_Topic`. The `DDS_DataWriter` and/or `DDS_DataReader` can copy this qos by using the operations `DDS_<DDS_Entity>_copy_from_topic_qos` and then `DDS_<DDS_Entity>_set_qos`. That way the application can relatively easily ensure the `QosPolicy` for the `DDS_Topic`, `DDS_DataReader` and `DDS_DataWriter` are consistent.

3.1.3.9 DDS_LifespanQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_LifespanQosPolicy
{ DDS_Duration_t duration; };
```

Description

This `QosPolicy` specifies the duration of the validity of the data written by the `DDS_DataWriter`.

Attributes

DDS_Duration_t duration - specifies the length in time of the validity of the data.

Detailed Description

This `QosPolicy` specifies the duration of the validity of the data written by the `DDS_DataWriter`. When this time has expired, the data will be removed or if it has not been delivered yet, it will not be delivered at all. In other words, the *duration* is the time in which the data is still valid. This means that during this period a `DDS_DataReader` can access the data or if the data has not been delivered yet, it still will be delivered. The default value of the *duration* is `DDS_DURATION_INFINITE`, indicating that the data does not expire.

This `QosPolicy` is applicable to a `DDS_DataWriter` and a `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` may be changed by using the `DDS_<DDS_Entity>_set_qos` operation.

Requested/Offered

The setting of this `QosPolicy` is only applicable to the publishing side, in other words the Requested/Offered constraints are not applicable. The communication will not be rejected on account of this `QosPolicy`. The notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side will not be raised.

DDS_TopicQos

This `QosPolicy` can be set on a `DDS_Topic`. The `DDS_DataWriter` and/or `DDS_DataReader` can copy this qos by using the operations `DDS_<DDS_Entity>_copy_from_topic_qos` and then `DDS_<DDS_Entity>_set_qos`. That way the application can relatively easily ensure the `QosPolicy` for the `DDS_Topic`, `DDS_DataReader` and `DDS_DataWriter` are consistent.

3.1.3.10 DDS_LivelinessQosPolicy

Synopsis

```
#include <dds_dcps.h>
enum DDS_LivelinessQosPolicyKind
{
    DDS_AUTOMATIC_LIVELINESS_QOS,
    DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS,
    DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS };
struct DDS_LivelinessQosPolicy
{
    DDS_LivelinessQosPolicyKind kind;
    DDS_Duration_t lease_duration; };
```

Description

This `QosPolicy` controls the way the liveliness of an `DDS_Entity` is being reported.

Attributes

DDS_LivelinessQosPolicyKind kind - controls the way the liveliness of an `DDS_Entity` is reported.

DDS_Duration_t lease_duration - specifies the duration of the interval within which the liveliness must be reported.

Detailed Description

This `QosPolicy` controls the way the liveliness of an `DDS_Entity` is being determined. The liveliness must be reported periodically before the `lease_duration` expires.

This `QosPolicy` is applicable to a `DDS_DataReader`, a `DDS_DataWriter` and a `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` cannot be changed any more.

Attributes

The `QosPolicy` is controlled by the attribute `kind` which can be:

- *DDS_AUTOMATIC_LIVELINESS_QOS* - the Data Distribution Service will take care of reporting the Liveliness automatically with a rate determined by the `lease_duration`
- *DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS* - the application must take care of reporting the liveliness before the `lease_duration` expires. If a `DDS_Entity` reports its liveliness, all `DDS_Entities` within the same `DDS_DomainParticipant` that have their liveliness kind set to `DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS`, can be considered alive

by the Data Distribution Service. Liveliness can be reported explicitly by calling the operation `DDS_DomainParticipant_assert_liveliness` or implicitly by writing some data

- `DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS` - the application must take care of reporting the liveliness before the `lease_duration` expires. This can explicitly be done by calling the operation `DDS_DataWriter_assert_liveliness` or implicitly by writing some data

The `lease_duration` specifies the duration of the interval within which the liveliness should be reported.

Requested/Offered

In case the Requested/Offered `QoSPolicy` are incompatible, the notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side and `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side is raised.

Table 11 DDS_LivelinessQoSPolicy

Requester Offered	AUTOMATIC	MANUAL_BY_ PARTICIPANT	MANUAL_BY_ TOPIC
AUTOMATIC	compatible	Incompatible	Incompatible
MANUAL_BY_PARTICIPANT	compatible	compatible	Incompatible
MANUAL_BY_TOPIC	compatible	compatible	compatible

Whether communication is established, is controlled by the Data Distribution Service, depending on the Requested/Offered `QoSPolicy` of the `DDS_DataWriter` and `DDS_DataReader`. In other words, the communication between any `DDS_DataWriter` and `DDS_DataReader` depends on what is expected by the `DDS_DataReader`. As a consequence, a `DDS_DataWriter` that has an incompatible `QoS` with respect to what a `DDS_DataReader` specified is not allowed to send its data to that specific `DDS_DataReader`. A `DDS_DataReader` that has an incompatible `QoS` with respect to what a `DDS_DataWriter` specified does not get any data from that particular `DDS_DataWriter`.

DDS_TopicQos

This `QoSPolicy` can be set on a `DDS_Topic`. The `DDS_DataWriter` and/or `DDS_DataReader` can copy this `qos` by using the operations `DDS_<DDS_Entity>_copy_from_topic_qos` and then `DDS_<DDS_Entity>_set_qos`. That way the application can relatively easily ensure the `QoSPolicy` for the `DDS_Topic`, `DDS_DataReader` and `DDS_DataWriter` are consistent.

3.1.3.11 DDS_OwnershipQosPolicy

Synopsis

```
#include <dds_dcps.h>
enum DDS_OwnershipQosPolicyKind
{ DDS_SHARED_OWNERSHIP_QOS,
  DDS_EXCLUSIVE_OWNERSHIP_QOS };
struct DDS_OwnershipQosPolicy
{ DDS_OwnershipQosPolicyKind kind; };
```

Description

This QosPolicy specifies whether a DDS_DataWriter exclusively owns an instance.

Attributes

DDS_OwnershipQosPolicyKind kind - specifies whether a DDS_DataWriter exclusively owns an instance.

Detailed Description

This QosPolicy specifies whether a DDS_DataWriter exclusively may own an instance. In other words, whether multiple DDS_DataWriter objects can write the same instance at the same time. The DDS_DataReader objects will only read the modifications on an instance from the DDS_DataWriter owning the instance.

Exclusive ownership is on an instance-by-instance basis. That is, a DDS_Subscriber can receive values written by a lower strength DDS_DataWriter as long as they affect instances whose values have not been written or registered by a higher-strength DDS_DataWriter.

This QosPolicy is applicable to a DDS_DataReader, a DDS_DataWriter and a DDS_Topic. After enabling of the concerning DDS_Entity, this QosPolicy cannot be changed any more.

Attribute

The QosPolicy is controlled by the attribute kind which can be:

- *DDS_SHARED_OWNERSHIP_QOS* (default) - the same instance can be written by multiple DDS_DataWriter objects. All updates will be made available to the DDS_DataReader objects. In other words it does not have a specific owner
- *DDS_EXCLUSIVE_OWNERSHIP_QOS* - the instance will only be accepted from one DDS_DataWriter which is the only one whose modifications will be visible to the DDS_DataReader objects

Requested/Offered

In case the Requested/Offered QoSPolicy are incompatible, the notification OFFERED_INCOMPATIBLE_QOS status on the offering side and REQUESTED_INCOMPATIBLE_QOS status on the requesting side is raised.

Table 12 Requested/Offered DDS_OwnershipQoSPolicy

Requested Offered	SHARED	EXCLUSIVE
SHARED	compatible	Incompatible
EXCLUSIVE	Incompatible	compatible

Whether communication is established, is controlled by the Data Distribution Service, depending on the Requested/Offered QoSPolicy of the DDS_DataWriter and DDS_DataReader. The value of the OWNERSHIP kind offered must exactly match the one requested or else they are considered incompatible. As a consequence, a DDS_DataWriter that has an incompatible QoS with respect to what a DDS_DataReader specified is not allowed to send its data to that specific DDS_DataReader. A DDS_DataReader that has an incompatible QoS with respect to what a DDS_DataWriter specified does not get any data from that particular DDS_DataWriter.

Exclusive Ownership

The DDS_DataWriter with the highest DDS_OwnershipStrengthQoSPolicy value and being alive (depending on the DDS_LivelinessQoSPolicy) and which has not violated its DDS_DeadlineQoSPolicy contract with respect to the instance, will be considered the owner of the instance. Consequently, the ownership can change as a result of:

- a DDS_DataWriter in the system with a higher value of the DDS_OwnershipStrengthQoSPolicy modifies the instance
- a change in the DDS_OwnershipStrengthQoSPolicy value (becomes less) of the DDS_DataWriter owning the instance
- a change in the liveliness (becomes not alive) of the DDS_DataWriter owning the instance
- a deadline with respect to the instance that is missed by the DDS_DataWriter that owns the instance.

Time-line

Each DDS_DataReader may detect the change of ownership at a different time. In other words, at a particular point in time, the DDS_DataReader objects do not have a consistent picture of who owns each instance for that DDS_Topic. Outside this grey area in time all DDS_DataReader objects will consider the same DDS_DataWriter to be the owner.

If multiple DDS_DataWriter objects with the same DDS_OwnershipStrengthQosPolicy modify the same instance, all DDS_DataReader objects will make the same choice of the particular DDS_DataWriter that is the owner. The DDS_DataReader is also notified of this via a status change that is accessible by means of the Listener or DDS_Condition mechanisms.

Ownership of an Instance

DDS_DataWriter objects are not aware whether they own a particular instance. There is no error or notification given to a DDS_DataWriter that modifies an instance it does not currently own.

TopicQos

This QosPolicy can be set on a DDS_Topic. The DDS_DataWriter and/or DDS_DataReader can copy this qos by using the operations DDS_Publisher/Subscriber_copy_from_topic_qos and then DDS_DataWriter/DataReader_set_qos. That way the application can relatively easily ensure the QosPolicy for the DDS_Topic, DDS_DataReader and DDS_DataWriter are consistent

3.1.3.12 DDS_OwnershipStrengthQosPolicy**Synopsis**

```
#include <dds_dcps.h>
    struct DDS_OwnershipStrengthQosPolicy
    { DDS_long value; };
```

Description

This QosPolicy specifies the value of the ownership strength of a DDS_DataWriter used to determine the ownership of an instance.

Attributes

DDS_long value - specifies the ownership strength of the DDS_DataWriter.

Detailed Description

This `QosPolicy` specifies the value of the ownership strength of a `DDS_DataWriter` used to determine the ownership of an instance. This ownership is used to arbitrate among multiple `DDS_DataWriter` objects that attempt to modify the same instance. This `QosPolicy` only applies if the `DDS_OwnershipQosPolicy` is of kind `DDS_EXCLUSIVE_OWNERSHIP_QOS`. For more information, see `DDS_OwnershipQosPolicy`.

This `QosPolicy` is applicable to a `DDS_DataWriter` only. After enabling of the concerning `DDS_Entity`, this `QosPolicy` may be changed by using the `DDS_DataWriter_set_qos` operation. When changed, the ownership of the instances may change with it.

3.1.3.13 DDS_PartitionQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_PartitionQosPolicy
{ DDS_StringSeq name; };
```

Description

This `QosPolicy` specifies the logical partitions in which the `DDS_Subscribers` and `DDS_Publishers` are active.

Attributes

DDS_StringSeq name - holds the sequence of strings, which specifies the partitions

Detailed Description

This `QosPolicy` specifies the logical partitions inside the domain in which the `DDS_Subscribers` and `DDS_Publishers` are active. This `QosPolicy` is particularly used to create a separate subspace, like a real domain versus a simulation domain. A `DDS_Publisher` and/or `DDS_Subscriber` can participate in more than one partition. Each string in the sequence of strings *name* defines a partition name. A partition name may contain wildcards. Sharing a partition means that at least one of the partition names in the sequence matches. When none of the partition names match, it is not considered an “incompatible” QoS and does not trigger any listeners or conditions. It only means that no communication is established. The default value of the attribute is an empty (zero-sized) sequence. This is treated as a special value that matches the partition.

This `QosPolicy` is applicable to a `DDS_Publisher` and `DDS_Subscriber`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` may be changed by using the `DDS_<DDS_Entity>_set_qos` operation. When changed, it modifies the association of `DDS_DataReader` and `DDS_DataWriter` objects. It may establish new associations or break existing associations. By default, `DDS_DataWriter` and `DDS_DataReader` objects belonging to a `DDS_Publisher` or `DDS_Subscriber` that do not specify a `DDS_PartitionQosPolicy`, will participate in the default partition. In this case the partition name is "".

Requested/Offered

The offered setting of this `QosPolicy` is independent of the one requested, in other words they are never considered incompatible. The communication will not be rejected on account of this `QosPolicy`. The notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side or `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side will not be raised.

3.1.3.14 DDS_PresentationQosPolicy

Synopsis

```
#include <dds_dcps.h>
enum DDS_PresentationQosPolicyAccessScopeKind
{
    DDS_INSTANCE_PRESENTATION_QOS,
    DDS_TOPIC_PRESENTATION_QOS,
    DDS_GROUP_PRESENTATION_QOS };
struct DDS_PresentationQosPolicy
{
    DDS_PresentationQosPolicyAccessScopeKind access_scope;
    DDS_boolean coherent_access;
    DDS_boolean ordered_access; };
```

Description

This `QosPolicy` controls the extent to which changes to data-instances can be made dependent on each other and also the kind of dependencies that can be propagated and maintained by the Data Distribution Service.

Attributes

PresentationQosPolicyAccessScopeKind access_scope - specifies the granularity of the changes that needs to be preserved when communicating a set of samples. Currently only the `DDS_INSTANCE_PRESENTATION_QOS` and `DDS_TOPIC_PRESENTATION_QOS` scopes are supported.

boolean coherent_access - controls whether the Data Distribution Service will preserve the groupings of changes, as indicated by the *access_scope*, made by a publishing application by means of the operations *begin_coherent_change* and *end_coherent_change*.

boolean ordered_access - controls whether the Data Distribution Service will preserve the order of the changes, as indicated by the *access_scope*. Currently only an *ordered_access* setting of *FALSE* is supported.

Detailed Description

The support for ‘coherent changes’ enables a publishing application to change the value of several data-instances that could belong to the same or different topics and have those changes be seen ‘atomically’ by the readers. This is useful in cases where the values are inter-related. For example, if there are two data-instances representing the ‘altitude’ and ‘velocity vector’ of the same aircraft and both are changed, it may be useful to communicate those values in a way the reader can see both together; otherwise it may erroneously interpret that the aircraft is on a collision course.

Basically this QosPolicy allows a Publisher to group a number of samples by enclosing them within calls to *DDS_Publisher_begin_coherent_change* and *DDS_Publisher_end_coherent_change* and treat them as if they are to be communicated as a single message. That is, the receiver will only be able to access the data after all the modifications in the set are available at the receiver end.

A connectivity change may occur in the middle of a set of coherent changes; for example, the set of partitions used by the Publisher or one of its Subscribers may change, a late-joining DataReader may appear on the network, or a communication failure may occur. In the event that such a change prevents an entity from receiving the entire set of coherent changes, that entity must behave as if it had received none of the set.

This QosPolicy is applicable to a Publisher and Subscriber. After enabling of the concerning Entity, this QosPolicy cannot be changed any more.

Attributes

The *PresentationQosPolicy* is applicable to both Publisher and Subscriber, but behaves differently on the publishing side and the subscribing side. The setting of *coherent_access* on a *DDS_Publisher* controls whether that Publisher will preserve the coherency of changes (enclosed by calls to *DDS_Publisher_begin_coherent_change* and *end_coherent_change*), as indicated by its *DDS_Publisher_access_scope* and as made available by its embedded DataWriters. However, the Subscriber settings determine whether a coherent set of samples will actually be delivered to the subscribing application in a coherent way.

- If a Publisher or Subscriber sets `coherent_access` to `FALSE`, it indicates that it does not want to maintain coherency between the different samples in a set: a Subscriber that receives only a part of this set may still deliver this partial set of samples to its embedded DataReaders.
- If both Publisher and Subscriber set `coherent_access` to `TRUE`, they indicate that they want to maintain coherency between the different samples in a set: a Subscriber that receives only a part of this set may not deliver this partial set of samples to its embedded DataReaders; it needs to wait for the set to become complete, and it will flush this partial set when it concludes that it will never be able to complete it.

Coherency is implemented on top of a transaction mechanism between individual DataWriters and DataReaders; completeness of a coherent set is determined by the successful completion of each of its participating transactions. The value of the `access_scope` attribute determines which combination of transactions constitute the contents of a coherent set.

The setting of `ordered_access` has no impact on the way in which a Publisher transmits its samples (although it does influence the RxO properties of this Publisher), but basically it determines whether a Subscriber will preserve the ordering of samples when the subscribing application uses its embedded DataReaders to read or take samples:

- If a Subscriber sets `ordered_access` to `FALSE`, it indicates that it does not want to maintain ordering between the different samples it receives: a subscribing application that reads or takes samples will receive these samples ordered by their key-values, which does probably not resemble the order they were written in.
- If a Subscriber sets `ordered_access` to `TRUE` (currently not supported), it indicates that it does want to maintain ordering within the specified `access_scope` between the different samples it receives: a subscribing application that reads or takes samples will receive these samples sorted by the order in which they were written..

The `access_scope` determines the maximum extent of coherent and/or ordered changes:

- If `access_scope` is set to `DDS_INSTANCE_PRESENTATION_QOS` and `coherent_access` is set to `TRUE`, then the Subscriber will behave, with respect to maintaining coherency, in a way similar to an `access_scope` that is set to `DDS_TOPIC_PRESENTATION_QOS`. This is caused by the fact that coherency is defined as the successful completion of all participating transactions. If a DataWriter writes a transaction containing samples from different instances, and a connected DataReader misses one of these samples, then the transaction failed and the coherent set is considered incomplete by the receiving DataReader. It doesn't matter that all the other instances have received their samples

successfully; an unsuccessful transaction by definition results in an incomplete coherent set. In that respect the DDS can offer no granularity that is more fine-grained with respect to coherency than that described by the `DDS_TOPIC_PRESENTATION_QOS`.

If `access_scope` is set to `DDS_INSTANCE_PRESENTATION_QOS` and `ordered_access` is set to `TRUE`, then the subscriber will maintain ordering between samples belonging to the same instance. Samples belonging to different instances will still be grouped by their key-values instead of by the order in which they were received.

- If `access_scope` is set to `DDS_TOPIC_PRESENTATION_QOS` and `coherent_access` is set to `TRUE`, then the DDS will define the scope of a coherent set on individual transactions. So a coherent set that spans samples coming from multiple DataWriters (indicated by its enclosure within calls to `DDS_Publisher_begin_coherent_change` and `DDS_Publisher_end_coherent_change` on their shared Publisher), is chopped up into separate and disjunct transactions (one for each participating DataWriter), where each transaction is processed separately. On the subscribing side this may result in the successful completion of some of these transactions, and the unsuccessful completion of some others. In such cases all DataReaders that received successful transactions will deliver the embedded content to their applications, without waiting for the completion of other transactions in other DataReaders connected to the same Subscriber.

If `access_scope` is set to `DDS_TOPIC_PRESENTATION_QOS` and `ordered_access` is set to `TRUE` (currently not supported), then the subscriber will maintain ordering between samples belonging to the same DataReader. This means that samples belonging to the same instance in the same DataReader may no longer be received consecutively if samples belonging to different instances were written in between.

- If `access_scope` is set to `DDS_GROUP_PRESENTATION_QOS` (currently not supported) and `coherent_access` is set to `TRUE`, then the DDS will define the scope of a coherent set on the sum of all participating transactions. So a coherent set that spans samples coming from multiple DataWriters (indicated by its enclosure within calls to `DDS_Publisher_begin_coherent_change` and `DDS_Publisher_end_coherent_change` on their shared Publisher), is chopped up into separate and disjunct transactions (one for each participating DataWriter), where each transactions is processed separately. On the subscribing side this may result in the successful completion of some of these transactions, and the unsuccessful completion of some others. However, each DataReader is only allowed to deliver the embedded content when all participating transactions completed successfully. This means that DataReaders that received successful transactions will need to wait for all other DataReaders attached to the same

Subscriber to also complete their transactions successfully. If one or more DataReaders conclude that they will not be able to complete their transactions successfully, then all DataReaders that participate in the original coherent set will flush the content of their transactions.

If `access_scope` is set to `DDS_GROUP_PRESENTATION_QOS` (currently not supported) and `ordered_access` is set to `TRUE` (currently not supported), then ordering is maintained between samples that are written by DataWriters attached to a common Publisher and received by DataReaders attached to a common Subscriber. This way the subscribing application can access the changes as a unit and/or in the proper order. However, this does not necessarily imply that the subscribing application will indeed access the changes as a unit and/or in the correct order. For that to occur, the subscribing application must use the proper logic in accessing its datareaders:

- Upon notification by the callback operation `on_data_on_readers` of the SubscriberListener or when triggered by the similar `DDS_DATA_ON_READERS` status of the Subscriber's `DDS_StatusCondition`, the application uses `DDS_Subscriber_begin_access` on the Subscriber to indicate it will be accessing data through the Subscriber. This will lock the embedded datareaders for any incoming messages during the coherent data access.
- Then it calls `DDS_Subscriber_get_datareaders` on the Subscriber to get the list of DataReader objects where data samples are available. Note that when `ordered_access` is `TRUE`, then the list of DataReaders may contain the same reader several times. In this manner the correct sample order can be maintained among samples in different DataReader objects.
- Following this it calls `DDS_DataReader_read` or `DDS_DataReader_take` on each DataReader in the same order returned to access all the relevant changes in the DataReader. Note that when `ordered_access` is `TRUE`, you should only read or take *one* sample at a time.
- Once it has called `read` or `take` on all the readers, it calls `DDS_Subscriber_end_access` on the Subscriber. This will unlock the embedded datareaders again.

Requested/Offered

In case the Requested/Offered `QosPolicy` are incompatible, the notification `DDS_OFFERED_INCOMPATIBLE_QOS` status on the offering side and `DDS_REQUESTED_INCOMPATIBLE_QOS` status on the requesting side is raised.

Table 13 Requested/Offered PresentationQosPolicy

<div>Requested Offered</div>	INSTANCE	Topic	Group
instance	compatible	INcompatible	INcompatible
topic	compatible	compatible	INcompatible
group	compatible	compatible	compatible

The value offered is considered compatible with the value requested if and only if the following conditions are met:

1. The inequality “offered access_scope >= requested access_scope” evaluates to ‘TRUE’. For the purposes of this inequality, the values of PRESENTATION access_scope are considered ordered such that INSTANCE < TOPIC < GROUP.
2. Requested coherent_access is FALSE, or else both offered and requested coherent_access are TRUE.
3. Requested ordered_access is FALSE, or else both offered and requested ordered_access are TRUE.

In case the quality offered by the Publisher is better than the value requested by the Subscriber, the subscriber’s values determine the resulting behaviour for the subscribing application. In other words, the quality specified at the Subscriber site overrules the corresponding value at the Publisher site.

Consider the following scenario:

1. A Publisher publishes coherent sets with access_scope is GROUP and coherent_access is TRUE.
2. A Subscriber subscribes to these coherent sets with access_scope is TOPIC and coherent_access is TRUE.
3. The Publisher writes a coherent set consisting of 2 samples of Topic A, and 2 samples of Topic B.
4. During transmission, the first sample of Topic B gets lost.

According to the access_scope of the Publisher, the coherent set is incomplete and can therefore not be delivered. However, according to the access_scope of the Subscriber, coherency needs to be maintained on a per Reader/Writer pair basis so the samples for Topic A will be delivered upon arrival, but the samples for Topic B will not.

Basically, when both coherent_access and ordered_access are set to FALSE, then the access_scope serves no other purpose than to determine connectivity between Publishers and Subscribers.

An `access_scope` value of `DDS_GROUP_PRESENTATION_QOS` and/or an `ordered_access` value of `TRUE` are not yet supported. Setting any of these values in your `PresentationQosPolicy` will result in a `DDS_RETCODE_NOT_SUPPORTED`.

3.1.3.15 DDS_ReaderDataLifecycleQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_ReaderDataLifecycleQosPolicy
{
    DDS_Duration_t autopurge_nowriter_samples_delay;
    DDS_Duration_t autopurge_disposed_samples_delay;
};
```

Description

This `QosPolicy` specifies the maximum duration for which the `DDS_DataReader` will maintain information regarding a data instance for which the `instance_state` becomes either `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`.

Attributes

DDS_Duration_t autopurge_nowriter_samples_delay - specifies the duration for which the `DDS_DataReader` will maintain information regarding a data instance for which the `instance_state` becomes `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`. By default the duration value is `DDS_DURATION_INFINITE`. When the delay time has expired, the data instance is marked so that it can be purged in the next garbage collection sweep.

DDS_Duration_t autopurge_disposed_samples_delay - specifies the duration for which the `DDS_DataReader` will maintain information regarding a data instance for which the `instance_state` becomes `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`. By default the duration value is `DDS_DURATION_INFINITE`. When the delay time has expired, the data instance is marked so that it can be purged in the next garbage collection sweep.

Detailed Description

This `QosPolicy` specifies the maximum duration for which the `DDS_DataReader` will maintain information regarding a data instance for which the `instance_state` becomes either `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`. The `DDS_DataReader` manages resources for instances and samples of those instances. The amount of resources managed depends on other `QosPolicies` like the `DDS_HistoryQosPolicy` and the `DDS_ResourceLimitsQosPolicy`. The `DDS_DataReader` can only release resources for data instances for which all

samples have been taken and the `instance_state` has become `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`. If an application does not take the samples belonging to a data instance with such an `instance_state`, the `DDS_DataReader` will never be able to release the maintained resources. By means of this `QosPolicy` the application can instruct the `DDS_DataReader` to release all resources related to the concerning data instance after a specified duration.

This `QosPolicy` is applicable to a `DDS_DataReader` only. After enabling of the concerning `DDS_DataReader`, this `QosPolicy` can be changed using the `set_qos` operation.

3.1.3.16 DDS_ReliabilityQosPolicy

Synopsis

```
#include <dds_dcps.h>
enum DDS_ReliabilityQosPolicyKind
{
    DDS_BEST_EFFORT_RELIABILITY_QOS,
    DDS_RELIABLE_RELIABILITY_QOS };
struct DDS_ReliabilityQosPolicy
{
    DDS_ReliabilityQosPolicyKind kind;
    DDS_Duration_t max_blocking_time;
    DDS_boolean synchronous; };
```

Description

This `QosPolicy` controls the level of reliability of the data distribution offered or requested by the `DDS_DataWriters` and `DDS_DataReaders`.

Attributes

DDS_ReliabilityQosPolicyKind kind - specifies the type of reliability which may be `DDS_BEST_EFFORT_RELIABILITY_QOS` or `DDS_RELIABLE_RELIABILITY_QOS`.

DDS_Duration_t max_blocking_time - specifies the maximum time the `DDS_DataWriter_write` operation may block when the `DDS_DataWriter` does not have space to store the value written or when synchronous communication is specified and all expected acknowledgements are not yet received.

DDS_boolean synchronous - specifies whether a `DataWriter` should wait for acknowledgements by all connected `DataReaders` that also have set a synchronous `ReliabilityQosPolicy`.

Detailed Description

This `QosPolicy` controls the level of reliability of the data distribution requested by a `DDS_DataReader` or offered by a `DDS_DataWriter`. In other words, it controls whether data is allowed to get lost in transmission or not.

This `QosPolicy` is applicable to a `DDS_DataReader`, `DDS_DataWriter` and `DDS_Topic`. After enabling of the concerning `DDS_Entity`, this `QosPolicy` cannot be changed any more.

Attributes

The `QosPolicy` is controlled by the attribute kind which can be:

- `DDS_RELIABLE_RELIABILITY_QOS` - the Data Distribution Service will attempt to deliver all samples in the `DDS_DataWriters` history; arrival-checks are performed and data may get re-transmitted in case of lost data. In the steady-state (no modifications communicated via the `DDS_DataWriter`) the Data Distribution Service guarantees that all samples in the `DDS_DataWriter` history will eventually be delivered to the all `DDS_DataReader` objects. Outside the steady-state the `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy` determine how samples become part of the history and whether samples can be discarded from it. In this case also the `max_blocking_time` must be set
- `DDS_BEST_EFFORT_RELIABILITY_QOS` - the Data Distribution Service will only attempt to deliver the data; no arrival-checks are being performed and any lost data is not re-transmitted (non-reliable). Presumably new values for the samples are generated often enough by the application so that it is not necessary to resent or acknowledge any samples.

The effect of the attribute `max_blocking_time` depends on the setting of the `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy` and the synchronous setting of the `DDS_ReliabilityQosPolicy`. In case the `DDS_HistoryQosPolicy` kind is set to `DDS_KEEP_ALL_HISTORY_QOS`, the `DDS_DataWriter_write` operation on the `DDS_DataWriter` may block if the modification would cause one of the limits specified in the `DDS_ResourceLimitsQosPolicy` to be exceeded. Also in case the synchronous attribute value of the `ReliabilityQosPolicy` is set to `TRUE` on both sides of a pair of connected `DataWriters` and `DataReaders`, then the `DataWriter` will wait until all its connected synchronous `DataReaders` have acknowledged the data.

Under these circumstances, the `max_blocking_time` attribute of the `DDS_ReliabilityQosPolicy` configures the maximum duration the `DDS_DataWriter_write` operation may block.

Requested/Offered

In case the Requested/Offered QoSPolicy are incompatible, the notification DDS_OFFERED_INCOMPATIBLE_QOS status on the offering side and DDS_REQUESTED_INCOMPATIBLE_QOS status on the requesting side is raised.

Table 14 Requested/Offered DDS_ReliabilityQoSPolicy

Requested Offered	BEST_EFFORT	RELIABLE
BEST_EFFORT	compatible	Incompatible
RELIABLE	compatible	compatible

DDS TopicQos

This QoSPolicy can be set on a DDS_Topic. The DDS_DataWriter and/or DDS_DataReader can copy this qos by using the operations DDS_<DDS_Entity>_copy_from_topic_qos and then DDS_<DDS_Entity>_set_qos. That way the application can relatively easily ensure the QoSPolicy for the DDS_Topic, DDS_DataReader and DDS_DataWriter are consistent.

3.1.3.17 DDS_ResourceLimitsQoSPolicy**Synopsis**

```
#include <dds_dcps.h>
struct DDS_ResourceLimitsQoSPolicy
{
    DDS_long max_samples;
    DDS_long max_instances;
    DDS_long max_samples_per_instance; };
```

Description

This QoSPolicy will specify the maximum amount of resources, which can be used by a DDS_DataWriter or DDS_DataReader.

Attributes

DDS_long max_samples - specifies the maximum number of data samples for all instances for any single DDS_DataWriter (or DDS_DataReader). By default, DDS_LENGTH_UNLIMITED.

DDS_long max_instances - specifies the maximum number of instances for any single DDS_DataWriter (or DDS_DataReader). By default, DDS_LENGTH_UNLIMITED.

DDS_long max_samples_per_instance - specifies the maximum number of samples of any single instance for any single DDS_DataWriter (or DDS_DataReader). By default, DDS_LENGTH_UNLIMITED.

Detailed Description

This QosPolicy controls the maximum amount of resources that the Data Distribution Service can use in order to meet the requirements imposed by the application and other QosPolicy settings.

This QosPolicy is applicable to a DDS_DataReader, a DDS_DataWriter and a DDS_Topic. After enabling of the concerning DDS_Entity, this QosPolicy cannot be changed any more.

Requested/Offered

The value of the QosPolicy offered is independent of the one requested, in other words they are never considered incompatible. The communication will not be rejected on account of this QosPolicy. The notification DDS_OFFERED_INCOMPATIBLE_QOS status on the offering side or DDS_REQUESTED_INCOMPATIBLE_QOS status on the requesting side will not be raised.

Resource limits

If the DDS_DataWriter objects are publishing samples faster than they are taken by the DDS_DataReader objects, the Data Distribution Service will eventually hit against some of the QosPolicy-imposed resource limits. Note that this may occur when just a single DDS_DataReader cannot keep up with its corresponding DDS_DataWriter.

In case the DDS_HistoryQosPolicy is DDS_KEEP_LAST_HISTORY_QOS, the setting of DDS_ResourceLimitsQosPolicy max_samples_per_instance must be compatible with the DDS_HistoryQosPolicy depth. For these two QosPolicy settings to be compatible, they must verify that depth <= max_samples_per_instance.

DDS_TopicQos

This QosPolicy can be set on a DDS_Topic. The DDS_DataWriter and/or DDS_DataReader can copy this qos by using the operations DDS_<DDS_Entity>_copy_from_topic_qos and then DDS_<DDS_Entity>_set_qos. That way the application can relatively easily ensure the QosPolicy for the DDS_Topic, DDS_DataReader and DDS_DataWriter are consistent.

3.1.3.18 DDS_SchedulingQosPolicy

Scope

DDS

Synopsis

```
#include <dds_dcps.h>
enum DDS_SchedulingClassQosPolicyKind
{
    DDS_SCHEDULE_DEFAULT,
    DDS_SCHEDULE_TIMESHARING,
    DDS_SCHEDULE_REALTIME };
struct DDS_SchedulingClassQosPolicy
{
    DDS_SchedulingClassQosPolicyKind kind; };
enum DDS_SchedulingPriorityQosPolicyKind
{
    DDS_PRIORITY_RELATIVE,
    DDS_PRIORITY_ABSOLUTE };
struct DDS_SchedulingPriorityQosPolicy
{
    DDS_SchedulingPriorityQosPolicyKind kind; };
struct DDS_SchedulingQosPolicy
{
    DDS_SchedulingClassQosPolicy scheduling_class;
    DDS_SchedulingPriorityQosPolicy scheduling_priority_kind;
    DDS_long scheduling_priority; };
```

Description

This QosPolicy specifies the scheduling parameters that will be used for a thread that is spawned by the DDS_DomainParticipant.



Note that some scheduling parameters may not be supported by the underlying Operating System or that you may need special privileges to select particular settings.

Attributes

DDS_SchedulingClassQosPolicyKind scheduling_class.kind - specifies the scheduling class used by the Operating System, which may be `DDS_SCHEDULE_DEFAULT`, `DDS_SCHEDULE_TIMESHARING` or `DDS_SCHEDULE_REALTIME`. Threads can only be spawned within the scheduling classes that are supported by the underlying Operating System.

DDS_SchedulingPriorityQosPolicyKind scheduling_priority_kind.kind - specifies the priority type, which may be either `DDS_PRIORITY_RELATIVE` or `DDS_PRIORITY_ABSOLUTE`.

DDS_long scheduling_priority - specifies the priority that will be assigned to threads spawned by the DDS_DomainParticipant. Threads can only be spawned with priorities that are supported by the underlying Operating System.

Detailed Description



This `QosPolicy` specifies the scheduling parameters that will be used for threads spawned by the `DDS_DomainParticipant`. Note that some scheduling parameters may not be supported by the underlying Operating System or that you may need special privileges to select particular settings. Refer to the documentation of your OS for more details on this subject.

Although the behaviour of the `scheduling_class` is highly dependent on the underlying OS, in general when running in a `Timesharing` class your thread will need to regularly yield execution to other threads of equal priority. In a `Realtime` class, your thread normally runs until completion and can only be pre-empted by higher priority threads. Often, the highest range of priorities is not accessible through a `Timesharing` Class.

The `scheduling_priority_kind` determines whether the specified `scheduling_priority` should be interpreted as an absolute priority or whether it should be interpreted relative to the priority of its creator, in this case the priority of the thread that created the `DDS_DomainParticipant`.

3.1.3.19 DDS_TimeBasedFilterQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_TimeBasedFilterQosPolicy
{ DDS_Duration_t minimum_separation; };
```

NOTE: This `QosPolicy` is not yet implemented. It is scheduled for a future release.

3.1.3.20 DDS_TopicDataQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_TopicDataQosPolicy
{ DDS_sequence_octet value; };
```

Description

This `QosPolicy` allows the application to attach additional information to a `DDS_Topic` `DDS_Entity`. This information is distributed with the `DDS_BuiltinTopics`.

Attributes

DDS_sequence_octet value - a sequence of octets that holds the application topic data. By default, the sequence has length 0.

Detailed Description

This `QosPolicy` allows the application to attach additional information to a `DDS_Topic` Entity. This information is distributed with the `BuiltinTopic`. An application that discovers a new `DDS_Topic` entity, can use this information to add additional functionality. The `DDS_TopicDataQosPolicy` is changeable and updates of the `BuiltinTopic` instance must be expected. Note that the Data Distribution Service is not aware of the real structure of the topic data (the Data Distribution System handles it as an opaque type) and that the application is responsible for correct mapping on structural types for the specific platform.

3.1.3.21 DDS_TransportPriorityQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_TransportPriorityQosPolicy
{ DDS_long value; };
```

Description

This `QosPolicy` specifies the priority with which the Data Distribution System can handle the data produced by the `DDS_DataWriter`.

Attributes

DDS_long value - specifies the priority with which the Data Distribution System can handle the data produced by the `DDS_DataWriter`.

Detailed Description

This `QosPolicy` specifies the priority with which the Data Distribution System can handle the data produced by a `DDS_DataWriter`. This `QosPolicy` is considered to be a hint to the Data Distribution Service to control the priorities of the underlying transport means. A higher value represents a higher priority and the full range of the type is supported. By default the transport priority is set to 0.

The `DDS_TransportPriorityQosPolicy` is applicable to both `DDS_Topic` and `DDS_DataWriter` entities. After enabling of the concerning `DDS_Entities`, this `QosPolicy` may be changed by using the `set_qos` operation.

TopicQos

Note that changing this `QosPolicy` for the `DDS_Topic` does not influence the behaviour of the Data Distribution System for existing `DDS_DataWriter` entities because this `QosPolicy` is only used by the operation `copy_from_topic_qos` and when specifying `DDS_DATAWRITER_QOS_USE_TOPIC_QOS` when creating the `DataWriter`.

3.1.3.22 DDS_UserDataQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_UserDataQosPolicy
{ DDS_sequence_octet value; };
```

Description

This `QosPolicy` allows the application to attach additional information to a `DDS_DomainParticipant`, `DDS_DataReader` or `DDS_DataWriter` `DDS_Entity`. This information is distributed with the Builtin Topics.

Attributes

DDS_sequence_octet value - is a sequence of octets that holds the application user data. By default, the sequence has length 0.

Detailed Description

This `QosPolicy` allows the application to attach additional information to a `DDS_DomainParticipant`, `DDS_DataReader` or `DDS_DataWriter` entity. This information is distributed with the Builtin Topics. An application that discovers a new `DDS_Entity` of the listed kind, can use this information to add additional functionality. The `DDS_UserDataQosPolicy` is changeable and updates of the Builtin Topic instance must be expected. Note that the Data Distribution Service is not aware of the real structure of the user data (the Data Distribution System handles it as an opaque type) and that the application is responsible for correct mapping on structural types for the specific platform.

3.1.3.23 DDS_WriterDataLifecycleQosPolicy

Synopsis

```
#include <dds_dcps.h>
struct DDS_WriterDataLifecycleQosPolicy
{ DDS_boolean autodispose_unregistered_instances; };
```

Description

This `QosPolicy` specifies whether the Data Distribution Service should automatically dispose instances that are unregistered by the `DDS_DataWriter`.

Attributes

DDS_boolean autodispose_unregistered_instances - specifies whether the Data Distribution Service should automatically dispose instances that are unregistered by this `DDS_DataWriter`.

Detailed Description

This `QosPolicy` controls the behaviour of the `DDS_DataWriter` with regards to the lifecycle of the data-instances it manages, that is, the data-instances that have been registered either explicitly using one of the `register` operations or implicitly by directly writing the data using the special `DDS_HANDLE_NIL` parameter. (See also Section 3.4.2.50, *SPACE_FooDataWriter_register_instance*, on page 316).

The `autodispose_unregistered_instances` flag controls what happens when an instance gets unregistered by the `DDS_DataWriter`:

- If the `DDS_DataWriter` unregisters the instance explicitly using either `SPACE_FooDataWriter_unregister_instance` or `SPACE_FooDataWriter_unregister_instance_w_timestamp`, then the `autodispose_unregistered_instances` flag is currently ignored and the instance is never disposed automatically.
- If the `DDS_DataWriter` unregisters its instances implicitly because it is deleted or if a `DDS_DataReader` detects a loss of liveliness of a connected `DDS_DataWriter`, then the `auto_dispose_unregistered_instances` flag determines whether the concerned instances are automatically disposed (`TRUE`) or not (`FALSE`).

The default value for the `autodispose_unregistered_instances` flag is `TRUE`. For `TRANSIENT` and `PERSISTENT` topics this means that all instances that are not explicitly unregistered by the application will by default be removed from the Transient and Persistent stores when the `DataWriter` is deleted or when a loss of its liveliness is detected.

3.1.4 DDS_Listener interface

This interface is the abstract base interface for all `Listener` interfaces. `Listeners` provide a generic mechanism for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a `QosPolicy` setting, etc. Each DCPS `DDS_Entity` supports its own specialized kind of `Listener`. `Listeners` are related to changes in communication status. For each `DDS_Entity` type, one specific `Listener` is derived from this interface. In the following modules, the following `Listeners` are derived from this interface:

- `DDS_DomainParticipantListener`
- `DDS_TopicListener`
- `DDS_PublisherListener`
- `DDS_DataWriterListener`

- DDS_SubscriberListener
- DDS_DataReaderListener.

The DDS_Entity type specific Listener interfaces are part of the application which must implement the interface operations. A user-defined class for these operations must be provided by the application which must extend from the **specific** Listener class. **All** Listener operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.

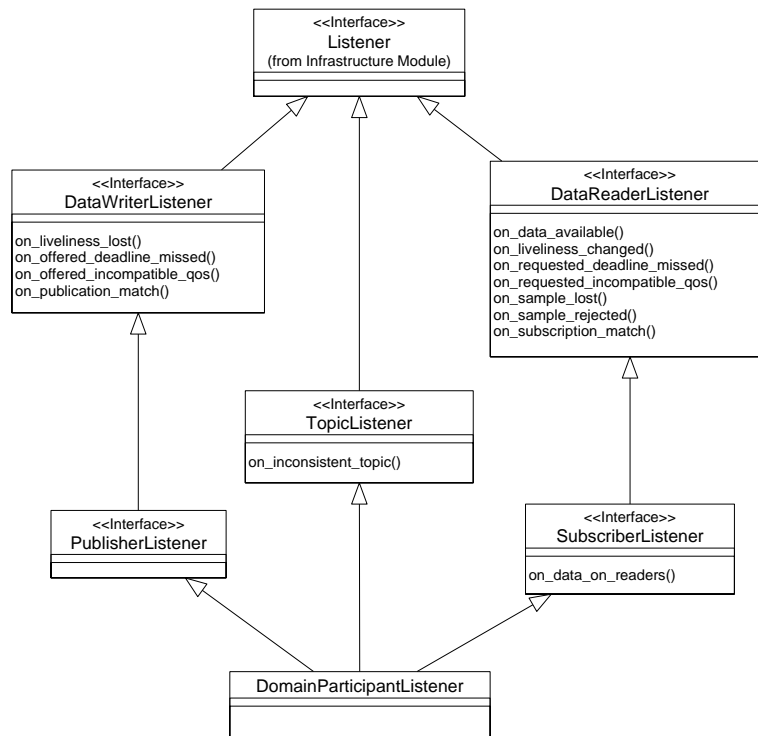


Figure 11 DCPS Listeners

The base class DDS_Listener does not contain any operations.

3.1.5 Struct DDS_Status

Each concrete DDS_Entity class has a set of DDS_Status attributes and for each attribute the DDS_Entity class provides an operation to read the value. Changes to DDS_Status attributes will affect associated DDS_StatusCondition and (invoked and associated) Listener objects.

The communication statuses whose changes can be communicated to the application depend on the `DDS_Entity`. The following table shows the relevant statuses for each `DDS_Entity`.

Table 15 Status Description Per `DDS_Entity`

DDS_Entity	Status Name	Meaning
DDS_Topic	DDS_INCONSISTENT_TOPIC_STATUS	Another <code>DDS_Topic</code> exists with the same name but with different characteristics.
DDS_Subscriber	DDS_DATA_ON_READERS_STATUS	New information is available.
DDS_DataReader	DDS_SAMPLE_REJECTED_STATUS	A (received) sample has been rejected.
	DDS_LIVELINESS_CHANGED_STATUS	The liveliness of one or more <code>DDS_DataWriter</code> objects that were writing instances read through the <code>DDS_DataReader</code> has changed. Some <code>DDS_DataWriter</code> have become “alive” or “not alive”.
	DDS_REQUESTED_DEADLINE_MISSED_STATUS	The deadline that the <code>DDS_DataReader</code> was expecting through its <code>DDS_DeadlineQoSPolicy</code> was not respected for a specific instance.
	DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS	A <code>QoSPolicy</code> setting was incompatible with what is offered.
	DDS_DATA_AVAILABLE_STATUS	New information is available.
	DDS_SAMPLE_LOST_STATUS	A sample has been lost (never received).
	DDS_SUBSCRIPTION_MATCHED_STATUS	The <code>DDS_DataReader</code> has found a <code>DDS_DataWriter</code> that matches the <code>DDS_Topic</code> and has compatible <code>QoS</code> .

Table 15 Status Description Per DDS_Entity (Continued)

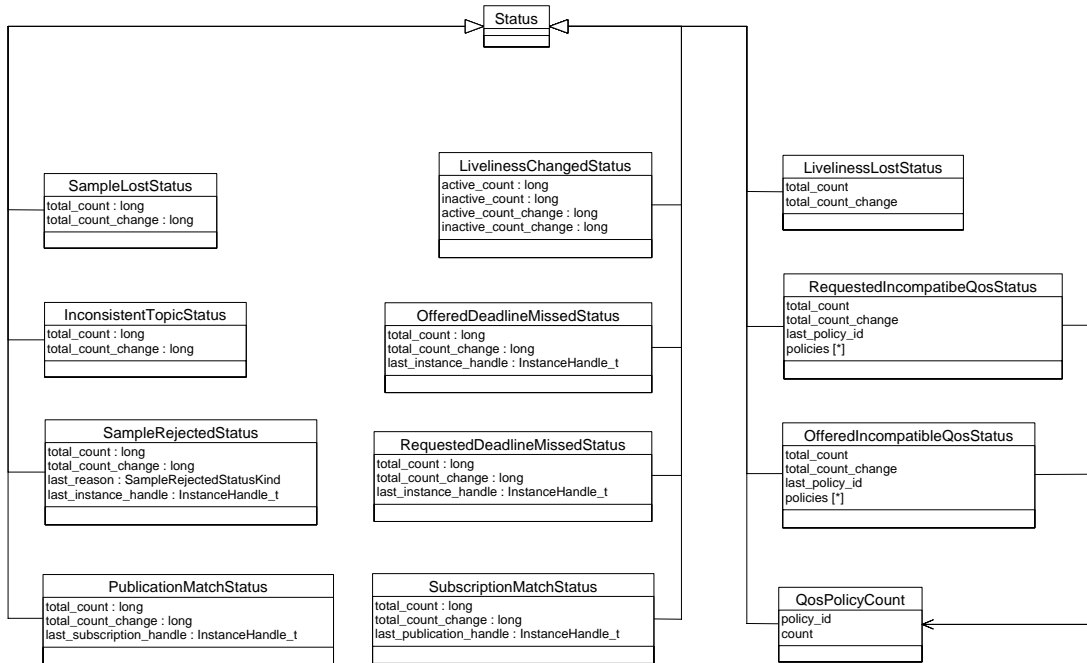
DDS_Entity	Status Name	Meaning
DDS_DataWriter	DDS_LIVELINESS_LOST_STATUS	The liveliness that the DDS_DataWriter has committed through its DDS_LivelinessQosPolicy was not respected; thus DDS_DataReader objects will consider the DDS_DataWriter as no longer “alive”.
	DDS_OFFERED_DEADLINE_MISSED_STATUS	The deadline that the DDS_DataWriter has committed through its DDS_DeadlineQosPolicy was not respected for a specific instance.
	DDS_OFFERED_INCOMPATIBLE_QOS_STATUS	A QosPolicy setting was incompatible with what was requested.
	DDS_PUBLICATION_MATCHED_STATUS	The DDS_DataWriter has found a DDS_DataReader that matches the DDS_Topic and has compatible QoS.

A `DDS_Status` attribute can be retrieved with the operation `get_<status_name>_status`. For example, to get the `DDS_InconsistentTopicStatus` value, the application must call the operation `DDS_Topic_get_inconsistent_topic_status`.

Conceptually associated with each `DDS_Entity` communication status is a logical `StatusChangedFlag`. This flag indicates whether that particular communication status has changed. The `StatusChangedFlag` is only conceptual, therefore, it is not important whether this flag actually exists.

For the plain communication `DDS_Status`, the `StatusChangedFlag` is initially set to `FALSE`. It becomes `TRUE` whenever the plain communication `DDS_Status` changes and it is reset to `FALSE` each time the application accesses the plain communication `DDS_Status` via the proper `get_<status_name>_status` operation on the `DDS_Entity`.

A flag set means that a change has occurred since the last time the application has read its value.

**Figure 12 DCPS DDS_Status Values**

Each DDS_Status attribute is implemented as a struct and therefore does not provide any operations. The interface description of these structs is as follows:

```

/*
 * struct DDS_<name>Status
 */
struct DDS_InconsistentTopicStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
};
struct DDS_SampleLostStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
};
enum DDS_SampleRejectedStatusKind
{
    DDS_NOT_REJECTED,
    DDS_REJECTED_BY_INSTANCES_LIMIT,
    DDS_REJECTED_BY_SAMPLES_LIMIT,
    DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT
};
struct DDS_SampleRejectedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_SampleRejectedStatusKind last_reason;
    DDS_InstanceHandle_t last_instance_handle;
};
struct DDS_LivelinessLostStatus
{
    DDS_long total_count;
};

```

```

        DDS_long total_count_change; };
struct DDS_LivelinessChangedStatus
{
    DDS_long alive_count;
    DDS_long not_alive_count;
    DDS_long alive_count_change;
    DDS_long not_alive_count_change;
    DDS_InstanceHandle_t last_publication_handle; };
struct DDS_OfferedDeadlineMissedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_InstanceHandle_t last_instance_handle; };
struct DDS_RequestedDeadlineMissedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_InstanceHandle_t last_instance_handle; };
struct DDS_OfferedIncompatibleQosStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_QosPolicyId_t last_policy_id;
    DDS_QosPolicyCountSeq policies; };
struct DDS_RequestedIncompatibleQosStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_QosPolicyId_t last_policy_id;
    DDS_QosPolicyCountSeq policies; };
struct DDS_PublicationMatchedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_long current_count;
    DDS_long current_count_change;
    DDS_InstanceHandle_t last_subscription_handle; };
struct DDS_SubscriptionMatchedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_long current_count;
    DDS_long current_count_change;
    DDS_InstanceHandle_t last_publication_handle; };

/*
 * implemented API operations
 * <no operations>
 */

```

The sections describe the usage of each DDS_<name>Status struct.

3.1.5.1 DDS_InconsistentTopicStatus

Synopsis

```

#include <dds_dcps.h>
struct DDS_InconsistentTopicStatus
{
    DDS_long total_count;

```

```
DDS_long total_count_change; };
```

Description

This struct contains the statistics about attempts to create other `DDS_Topic`s with the same name but with different characteristics.

Attributes

DDS_long total_count - the total detected cumulative count of `DDS_Topic` creations, whose name matches the `DDS_Topic` to which this `DDS_Status` is attached and whose characteristics are inconsistent.

DDS_long total_count_change - the change in `total_count` since the last time the `Listener` was called or the `DDS_Status` was read.

Detailed Description

This struct contains the statistics about attempts to create other `DDS_Topic`s with the same name but with different characteristics.

The attribute `total_count` holds the total cumulative count of `DDS_Topic` creations, whose name matches the `DDS_Topic` to which this `DDS_Status` is attached and whose characteristics are inconsistent.

The attribute `total_count_change` holds the incremental number of inconsistent `DDS_Topic`s, since the last time the `Listener` was called or the `DDS_Status` was read.

3.1.5.2 DDS_LivelinessChangedStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_LivelinessChangedStatus
{
    DDS_long alive_count;
    DDS_long not_alive_count;
    DDS_long alive_count_change;
    DDS_long not_alive_count_change;
    DDS_InstanceHandle_t last_publication_handle; };
```

Description

This struct contains the statistics about whether the liveliness of one or more connected `DDS_DataWriter` objects has changed.

Attributes

DDS_long alive_count - the total count of currently alive `DDS_DataWriter` objects that write the topic read by the `DDS_DataReader` to which this `DDS_Status` is attached.

DDS_long not_alive_count - the total count of currently not alive DDS_DataWriter objects that wrote the topic read by the DDS_DataReader to which this DDS_Status is attached.

DDS_long alive_count_change - the change in alive_count since the last time the Listener was called or the DDS_Status was read.

DDS_long not_alive_count_change - the change in not_alive_count since the last time the Listener was called or the DDS_Status was read.

DDS_InstanceHandle_t last_publication_handle - handle to the last DDS_DataWriter whose change in liveliness caused this status to change.

Detailed Description

This struct contains the statistics about whether the liveliness of one or more connected DDS_DataWriter objects that were writing instances read through the DDS_DataReader has changed. In other words, some DDS_DataWriters have become “alive” or “not alive”.

The attribute *alive_count* holds the total number of currently alive DDS_DataWriter objects that write the topic read by the DDS_DataReader to which this DDS_Status is attached. This count increases when a newly matched DDS_DataWriter asserts its liveliness for the first time or when a DDS_DataWriter previously considered to be not alive reasserts its liveliness. The count decreases when a DDS_DataWriter considered alive fails to assert its liveliness and becomes not alive, whether because it was deleted normally or for some other reason.

The attribute *not_alive_count* holds the total count of currently not alive DDS_DataWriters that wrote the topic read by the DDS_DataReader to which this DDS_Status is attached, and that are no longer asserting their liveliness. This count increases when a DDS_DataWriter considered alive fails to assert its liveliness and becomes not alive for some reason other than the normal deletion of that DDS_DataWriter. It decreases when a previously not alive DDS_DataWriter either reasserts its liveliness or is deleted normally.

The attribute *alive_count_change* holds the change in *alive_count* since the last time the Listener was called or the DDS_Status was read.

The attribute *not_alive_count_change* holds the change in *not_alive_count* since the last time the Listener was called or the DDS_Status was read.

The attribute *last_publication_handle* contains the instance handle to the *DDS_PublicationBuiltinTopicData* instance that represents the last datawriter whose change in liveliness caused this status to change. Be aware that this handle belongs to *another* datareader, the



DDS_PublicationBuiltinTopicDataDataReader in the builtin-subscriber, and has no meaning in the context of the datareader from which the *DDS_LivelinessChangedStatus* was obtained. If the builtin-subscriber has not explicitly been obtained using *DDS_DomainParticipant_get_builtin_subscriber*, then there is no *DDS_PublicationBuiltinTopicDataDataReader* as well, in which case the *last_publication_handle* will be set to *DDS_HANDLE_NIL*.

3.1.5.3 DDS_LivelinessLostStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_LivelinessLostStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
};
```

Description

This struct contains the statistics about whether the liveliness of the *DDS_DataWriter* to which this *DDS_Status* is attached has been committed through its *DDS_LivelinessQosPolicy*.

Attributes

DDS_long total_count - the total cumulative count of times the *DDS_DataWriter* to which this *DDS_Status* is attached failed to actively signal its liveliness within the offered liveliness period.

DDS_long total_count_change - the change in *total_count* since the last time the Listener was called or the *DDS_Status* was read.

Detailed Description

This struct contains the statistics about whether the liveliness of the *DDS_DataWriter* to which this *DDS_Status* is attached has been committed through its *DDS_LivelinessQosPolicy*. In other words, whether the *DDS_DataWriter* failed to actively signal its liveliness within the offered liveliness period. In such a case, the connected *DDS_DataReader* objects will consider the *DDS_DataWriter* as no longer “alive”.

The attribute *total_count* holds the total cumulative number of times that the previously-alive *DDS_DataWriter* became not alive due to a failure to actively signal its liveliness within its offered liveliness period. This count does not change when an already not alive *DDS_DataWriter* simply remains not alive for another liveliness period.

The attribute *total_count_change* holds the change in *total_count* since the last time the Listener was called or the *DDS_Status* was read.

3.1.5.4 DDS_OfferedDeadlineMissedStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_OfferedDeadlineMissedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_InstanceHandle_t last_instance_handle;
};
```

Description

This struct contains the statistics about whether the deadline that the DDS_DataWriter to which this DDS_Status is attached has committed through its DDS_DeadlineQosPolicy, was not respected for a specific instance.

Attributes

DDS_long total_count - the total cumulative count of times the DDS_DataWriter to which this DDS_Status is attached failed to write within its offered deadline.

DDS_long total_count_change - the change in total_count since the last time the Listener was called or the DDS_Status was read.

DDS_InstanceHandle_t last_instance_handle - the handle to the last instance in the DDS_DataWriter to which this DDS_Status is attached, for which an offered deadline was missed.

Detailed Description

This struct contains the statistics about whether the deadline that the DDS_DataWriter to which this DDS_Status is attached has committed through its DDS_DeadlineQosPolicy, was not respected for a specific instance.

The attribute total_count holds the total cumulative number of offered deadline periods elapsed during which the DDS_DataWriter to which this DDS_Status is attached failed to provide data. Missed deadlines accumulate; that is, each deadline period the total_count will be incremented by one.

The attribute total_count_change holds the change in total_count since the last time the Listener was called or the DDS_Status was read.

The attribute last_instance_handle holds the handle to the last instance in the DDS_DataWriter to which this DDS_Status is attached, for which an offered deadline was missed.

3.1.5.5 DDS_OfferedIncompatibleQosStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_OfferedIncompatibleQosStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    QosPolicyId_t last_policy_id;
    QosPolicyCountSeq policies; };
```

Description

This struct contains the statistics about whether an offered QosPolicy setting was incompatible with the requested QosPolicy setting.

Attributes

DDS_long total_count - the total cumulative count of DDS_DataReader objects discovered by the DDS_DataWriter with the same DDS_Topic and Partition and with a requested DDS_DataReaderQos that was incompatible with the one offered by the DDS_DataWriter.

DDS_long total_count_change - the change in total_count since the last time the Listener was called or the DDS_Status was read.

QosPolicyId_t last_policy_id - the id of one of the QosPolicy settings that was found to be incompatible with what was offered, the last time an incompatibility was detected.

QosPolicyCountSeq policies - a list containing for each QosPolicy the total number of times that the concerned DDS_DataWriter discovered a DDS_DataReader for the same DDS_Topic and a requested DDS_DataReaderQos that is incompatible with the one offered by the DDS_DataWriter.

Detailed Description

This struct contains the statistics about whether an offered QosPolicy setting was incompatible with the requested QosPolicy setting.

The Request/Offering mechanism is applicable between:

- the DDS_DataWriter and the DDS_DataReader. If the QosPolicy settings between DDS_DataWriter and DDS_DataReader are incompatible, no communication between them is established. In addition the DDS_DataWriter will be informed via a DDS_REQUESTED_INCOMPATIBLE_QOS status change and the DDS_DataReader will be informed via an DDS_OFFERED_INCOMPATIBLE_QOS status change.

- the `DDS_DataWriter` and the `Durability Service` (as a built-in `DDS_DataReader`). If the `QosPolicy` settings between `DDS_DataWriter` and the `Durability Service` are inconsistent, no communication between them is established. In that case data published by the `DDS_DataWriter` will not be maintained by the service and as a consequence will not be available for late joining `DDS_DataReaders`. The `QosPolicy` of the `Durability Service` in the role of `DDS_DataReader` is specified by the `DDS_DurabilityServiceQosPolicy` in the `DDS_Topic`.
- the `Durability Service` (as a built-in `DDS_DataWriter`) and the `DDS_DataReader`. If the `QosPolicy` settings between the `Durability Service` and the `DDS_DataReader` are inconsistent, no communication between them is established. In that case the `Durability Service` will not publish historical data to late joining `DDS_DataReaders`. The `QosPolicy` of the `Durability Service` in the role of `DDS_DataWriter` is specified by the `DDS_DurabilityServiceQosPolicy` in the `DDS_Topic`.

The attribute `total_count` holds the total cumulative count of `DDS_DataReader` objects discovered by the `DDS_DataWriter` with the same `DDS_Topic` and a requested `DDS_DataReaderQos` that was incompatible with the one offered by the `DDS_DataWriter`.

The attribute `total_count_change` holds the change in `total_count` since the last time the `Listener` was called or the `DDS_Status` was read.

The attribute `last_policy_id` holds the id of one of the `QosPolicy` settings that was found to be incompatible with what was offered, the last time an incompatibility was detected.

The attribute `policies` holds a list containing for each `QosPolicy` the total number of times that the concerned `DDS_DataWriter` discovered an incompatible `DDS_DataReader` for the same `DDS_Topic`. Each element in the list represents a counter for a different `QosPolicy`, identified by a corresponding unique index number. A named list of all index numbers is expressed as a set of constants in the API. See *Table 16* below for an overview of all these constants.

Table 16 Overview of All Named QosPolicy Indexes

Index Name	Index Value
<code>DDS_INVALID_QOS_POLICY_ID</code>	0
<code>DDS_USERDATA_QOS_POLICY_ID</code>	1
<code>DDS_DURABILITY_QOS_POLICY_ID</code>	2
<code>DDS_PRESENTATION_QOS_POLICY_ID</code>	3
<code>DDS_DEADLINE_QOS_POLICY_ID</code>	4

Table 16 Overview of All Named QosPolicy Indexes

Index Name	Index Value
DDS_LATENCYBUDGET_QOS_POLICY_ID	5
DDS_OWNERSHIP_QOS_POLICY_ID	6
DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID	7
DDS_LIVELINESS_QOS_POLICY_ID	8
DDS_TIMEBASEDFILTER_QOS_POLICY_ID	9
DDS_PARTITION_QOS_POLICY_ID	10
DDS_RELIABILITY_QOS_POLICY_ID	11
DDS_DESTINATIONORDER_QOS_POLICY_ID	12
DDS_HISTORY_QOS_POLICY_ID	13
DDS_RESOURCELIMITS_QOS_POLICY_ID	14
DDS_ENTITYFACTORY_QOS_POLICY_ID	15
DDS_WRITERDATALIFECYCLE_QOS_POLICY_ID	16
DDS_READERDATALIFECYCLE_QOS_POLICY_ID	17
DDS_TOPICDATA_QOS_POLICY_ID	18
DDS_GROUPDATA_QOS_POLICY_ID	19
DDS_TRANSPORTPRIORITY_QOS_POLICY_ID	20
DDS_LIFESPAN_QOS_POLICY_ID	21
DDS_DURABILITYSERVICE_QOS_POLICY_ID	22

3.1.5.6 DDS_PublicationMatchedStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_PublicationMatchedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_long current_count;
    DDS_long current_count_change;
    DDS_InstanceHandle_t last_subscription_handle; };
```

Description

The functionality behind the DDS_PublicationMatchedStatus is not yet implemented. It is scheduled for a future release.

3.1.5.7 DDS_RequestedDeadlineMissedStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_RequestedDeadlineMissedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_InstanceHandle_t last_instance_handle;
};
```

Description

This struct contains the statistics about whether the deadline that the DDS_DataReader to which this DDS_Status is attached was expecting through its DDS_DeadlineQosPolicy, was not respected for a specific instance.

Attributes

DDS_long total_count - the total cumulative count of the missed deadlines detected for any instance read by the DDS_DataReader to which this DDS_Status is attached.

DDS_long total_count_change - the change in total_count since the last time the Listener was called or the DDS_Status was read.

DDS_InstanceHandle_t last_instance_handle - the handle to the last instance in the DDS_DataReader to which this DDS_Status is attached for which a missed deadline was detected.

Detailed Description

This struct contains the statistics about whether the deadline that the DDS_DataReader to which this DDS_Status is attached was expecting through its DDS_DeadlineQosPolicy, was not respected for a specific instance. Missed deadlines accumulate, that is, each deadline period the total_count will be incremented by one for each instance for which data was not received.

The attribute total_count holds the total cumulative count of the missed deadlines detected for any instance read by the DDS_DataReader.

The attribute total_count_change holds the change in total_count since the last time the Listener was called or the DDS_Status was read.

The attribute last_instance_handle holds the handle to the last instance in the DDS_DataReader for which a missed deadline was detected.

3.1.5.8 DDS_RequestedIncompatibleQosStatus

Synopsis

```
#include <dds_dcps.h>
```

```
struct DDS_RequestedIncompatibleQosStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    QosPolicyId_t last_policy_id;
    QosPolicyCountSeq policies; };
```

Description

This struct contains the statistics about whether a requested QosPolicy setting was incompatible with the offered QosPolicy setting.

Attributes

DDS_long total_count - holds the total cumulative count of DDS_DataWriter objects, discovered by the DDS_DataReader to which this DDS_Status is attached, with the same DDS_Topic and an offered DDS_DataWriterQos that was incompatible with the one requested by the DDS_DataReader.

DDS_long total_count_change - holds the change in total_count since the last time the Listener was called or the DDS_Status was read.

QosPolicyId_t last_policy_id - holds the DDS_<name>_QOS_POLICY_ID of one of the QosPolicies that was found to be incompatible with what was requested, the last time an incompatibility was detected.

QosPolicyCountSeq policies - a list containing (for each QosPolicy) the total number of times that the concerned DDS_DataReader discovered a DDS_DataWriter with the same DDS_Topic and an offered DDS_DataWriterQos that is incompatible with the one requested by the DDS_DataReader.

Detailed Description

This struct contains the statistics about whether a requested QosPolicy setting was incompatible with the offered QosPolicy setting.

The Request/Offering mechanism is applicable between:

- the DDS_DataWriter and the DDS_DataReader. If the QosPolicy settings between DDS_DataWriter and DDS_DataReader are incompatible, no communication between them is established. In addition the DDS_DataWriter will be informed via a DDS_REQUESTED_INCOMPATIBLE_QOS status change and the DDS_DataReader will be informed via an DDS_OFFERED_INCOMPATIBLE_QOS status change.
- the DDS_DataWriter and the Durability Service (as a built-in DDS_DataReader). If the QosPolicy settings between DDS_DataWriter and the Durability Service are inconsistent, no communication between them is established. In that case data published by the DDS_DataWriter will not be maintained by the service and as a consequence will not be available for late

joining DDS_DataReaders. The QosPolicy of the Durability Service in the role of DDS_DataReader is specified by the DDS_DurabilityServiceQosPolicy in the DDS_Topic.

- the Durability Service (as a built-in DDS_DataWriter) and the DDS_DataReader. If the QosPolicy settings between the Durability Service and the DDS_DataReader are inconsistent, no communication between them is established. In that case the Durability Service will not publish historical data to late joining DDS_DataReaders. The QosPolicy of the Durability Service in the role of DDS_DataWriter is specified by the DDS_DurabilityServiceQosPolicy in the DDS_Topic.

The attribute `total_count` holds the total cumulative count of DDS_DataWriter objects discovered by the DDS_DataReader with the same DDS_Topic and an offered DDS_DataWriterQos that was incompatible with the one requested by the DDS_DataReader.

The attribute `total_count_change` holds the change in `total_count` since the last time the Listener was called or the DDS_Status was read.

The attribute `last_policy_id` holds the DDS_<name>_QOS_POLICY_ID of one of the QosPolicies that was found to be incompatible with what was requested, the last time an incompatibility was detected.

The attribute `policies` holds a list containing for each QosPolicy: the total number of times that the concerned DDS_DataReader discovered an incompatible DDS_DataWriter for the same DDS_Topic. Each element in the list represents a counter for a different QosPolicy, identified by a corresponding unique index number. A named list of all index numbers is expressed as a set of constants in the API. See Table 16, *Overview of All Named QosPolicy Indexes*, on page 112 for an overview of all these constants.

3.1.5.9 DDS_SampleLostStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_SampleLostStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
};
```

Description

This struct contains the statistics about whether a sample has been lost (never received).

Attributes

DDS_long total_count - holds the total cumulative count of all samples lost across all instances of data published under the *DDS_Topic*.

DDS_long total_count_change - holds the change in *total_count* since the last time the Listener was called or the *DDS_Status* was read.

Detailed Description

This struct contains the statistics about whether a sample has been lost (never received). The status is independent of the differences in instances, in other words, it includes all samples lost across all instances of data published under the *DDS_Topic*.

total_count holds the total cumulative count of all samples lost across all instances of data published under the *DDS_Topic*.

total_count_change holds the change in *total_count* since the last time the Listener was called or the *DDS_Status* was read.

3.1.5.10 DDS_SampleRejectedStatus

Synopsis

```
#include <dds_dcps.h>

enum DDS_SampleRejectedStatusKind
{
    DDS_NOT_REJECTED,
    DDS_REJECTED_BY_INSTANCES_LIMIT,
    DDS_REJECTED_BY_SAMPLES_LIMIT,
    DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT };

struct DDS_SampleRejectedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_SampleRejectedStatusKind last_reason;
    DDS_InstanceHandle_t last_instance_handle; };

```

Description

This struct contains the statistics about samples that have been rejected.

Attributes

DDS_long total_count - holds the total cumulative count of samples rejected by the *DDS_DataReader* to which this *DDS_Status* is attached.

DDS_long total_count_change - holds the change in *total_count* since the last time the Listener was called or the *DDS_Status* was read.

DDS_SampleRejectedStatusKind last_reason - holds the reason for rejecting the last sample.

DDS_InstanceHandle_t last_instance_handle - holds the handle to the instance which would have updated by the last sample that was rejected.

Detailed Description

This struct contains the statistics about whether a received sample has been rejected.

The attribute *total_count* holds the total cumulative count of samples rejected by the *DDS_DataReader* to which this *DDS_Status* is attached.

The attribute *total_count_change* holds the change in *total_count* since the last time the *Listener* was called or the *DDS_Status* was read.

The attribute *last_reason* holds the reason for rejecting the last sample. The attribute can have the following values:

- *DDS_NOT_REJECTED* - no sample has been rejected yet.
- *DDS_REJECTED_BY_INSTANCES_LIMIT* - the sample was rejected because it would exceed the maximum number of instances set by the *DDS_ResourceLimitsQosPolicy*.
- *DDS_REJECTED_BY_SAMPLES_LIMIT* - the sample was rejected because it would exceed the maximum number of samples set by the *DDS_ResourceLimitsQosPolicy*.
- *DDS_REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT* - the sample was rejected because it would exceed the maximum number of samples per instance set by the *DDS_ResourceLimitsQosPolicy*.

The attribute *last_instance_handle* holds the handle to the instance which would have updated by the last sample that was rejected.

3.1.5.11 DDS_SubscriptionMatchedStatus

Synopsis

```
#include <dds_dcps.h>
struct DDS_SubscriptionMatchedStatus
{
    DDS_long total_count;
    DDS_long total_count_change;
    DDS_long current_count;
    DDS_long current_count_change;
    DDS_InstanceHandle_t last_publication_handle; };
```

Description

The functionality behind the *DDS_SubscriptionMatchedStatus* is not yet implemented. It is scheduled for a future release.

3.1.6 Class DDS_WaitSet

A DDS_WaitSet object allows an application to wait until one or more of the attached DDS_Condition objects evaluates to TRUE or until the timeout expires.

The DDS_WaitSet has no factory and must be created by the application. It is directly created as an object by using DDS_WaitSet constructors.

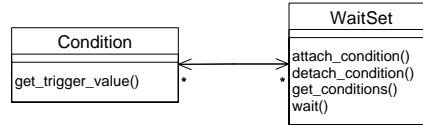


Figure 13 DCPS DDS_WaitSets

The interface description of this class is as follows:

```

/*
 * interface DDS_WaitSet
 */
/*
 * implemented API operations
 */
DDS_WaitSet
DDS_WaitSet__alloc
(void);
DDS_ReturnCode_t
DDS_WaitSet_wait
(DDS_WaitSet _this,
 DDS_ConditionSeq *active_conditions,
 const DDS_Duration_t *timeout);
DDS_ReturnCode_t
DDS_WaitSet_attach_condition
(DDS_WaitSet _this,
 const DDS_Condition cond);
DDS_ReturnCode_t
DDS_WaitSet_detach_condition
(DDS_WaitSet _this,
 const DDS_Condition cond);
DDS_ReturnCode_t
DDS_WaitSet_get_conditions
(DDS_WaitSet _this,
 DDS_ConditionSeq *attached_conditions);

```

The following sections describe the usage of all DDS_WaitSet operations.

3.1.6.1 DDS_WaitSet__alloc

Synopsis

```
#include <dds_dcps.h>
```

```
DDS_WaitSet
DDS_WaitSet__alloc
(void);
```

Description

This operation creates a new `DDS_WaitSet`.

Parameters

<none>

Return Value

DDS_WaitSet - handle to the newly created `DDS_WaitSet`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_WaitSet`. The `DDS_WaitSet` must be created using this operation. In other words, the application is not allowed to declare an object of type `DDS_WaitSet`. When the application wants to release the `DDS_WaitSet` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the `DDS_WaitSet`, a `DDS_OBJECT_NIL` pointer is returned instead.

3.1.6.2 DDS_WaitSet_attach_condition

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_WaitSet_attach_condition
(DDS_WaitSet _this,
 const DDS_Condition cond);
```

Description

This operation attaches a `DDS_Condition` to the `DDS_WaitSet`.

Parameters

in DDS_WaitSet _this - the `DDS_WaitSet` object on which the operation is operated.

in const DDS_Condition cond - a pointer to a `DDS_Condition`.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation attaches a DDS_Condition to the DDS_WaitSet. The parameter *cond* must be either a DDS_ReadCondition, DDS_QueryCondition, DDS_StatusCondition or DDS_GuardCondition. To get this parameter see:

- DDS_ReadCondition created by
 DDS_DataReader_create_readcondition
- DDS_QueryCondition created by
 DDS_DataReader_create_querycondition
- DDS_StatusCondition retrieved by
 DDS_<Entity>_get_statuscondition on an DDS_<Entity>
- DDS_GuardCondition created by the C operation
 DDS_GuardCondition__alloc.

When a DDS_GuardCondition is initially created, the *trigger_value* is FALSE.

When a DDS_Condition, whose *trigger_value* evaluates to TRUE, is attached to a DDS_WaitSet that is currently being waited on (using the DDS_WaitSet_wait operation), the DDS_WaitSet will unblock immediately.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_Condition is attached to the DDS_WaitSet.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *cond* is not a valid DDS_Condition.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.1.6.3 DDS_WaitSet_detach_condition

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
```

```
DDS_WaitSet_detach_condition
(DDS_WaitSet _this,
 const DDS_Condition cond)
```

Description

This operation detaches a DDS_Condition from the DDS_WaitSet.

Parameters

in DDS_WaitSet _this - the DDS_WaitSet object on which the operation is operated.

in const DDS_Condition cond - a pointer to a DDS_Condition in the DDS_WaitSet.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation detaches a DDS_Condition from the DDS_WaitSet. If the DDS_Condition was not attached to this DDS_WaitSet, the operation returns DDS_RETCODE_PRECONDITION_NOT_MET.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_Condition is detached from the DDS_WaitSet.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *cond* is not a valid DDS_Condition.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the DDS_Condition was not attached to this DDS_WaitSet.

3.1.6.4 DDS_WaitSet_get_conditions

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_WaitSet_get_conditions
        (DDS_WaitSet _this,
         DDS_ConditionSeq *attached_conditions);
```

Description

This operation retrieves the list of attached conditions.

Parameters

in DDS_WaitSet _this - the DDS_WaitSet object on which the operation is operated.

*inout DDS_ConditionSeq *attached_conditions* - the inout parameter *attached_conditions* is a sequence, which is used to pass the list of attached conditions.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation retrieves the list of attached conditions in the DDS_WaitSet. The parameter *attached_conditions* is a pointer to a sequence which afterwards will point to the sequence of attached conditions. The *attached_conditions* sequence and its buffer may be pre-allocated by the application and therefore must either be re-used in a subsequent invocation of the DDS_WaitSet_get_conditions operation or be released by calling DDS_free on the returned *attached_conditions*. If the pre-allocated sequence is not big enough to hold the number of attached DDS_Conditions, the sequence will automatically be (re-)allocated to fit the required size. The resulting sequence will either be an empty sequence, meaning there were no conditions attached, or will contain a list of DDS_ReadCondition, DDS_QueryCondition, DDS_StatusCondition and DDS_GuardCondition. These conditions previously have been attached by DDS_WaitSet_attach_condition and were created by there respective create operation:

- DDS_ReadCondition created by
 DDS_DataReader_create_readcondition

- `DDS_QueryCondition` created by `DDS_DataReader_create_querycondition`
- `DDS_StatusCondition` retrieved by `DDS_<Entity>_get_statuscondition` on an `DDS_<Entity>`
- `DDS_GuardCondition` created by the C operation `DDS_GuardCondition__alloc`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the list of attached conditions is returned
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.1.6.5 `DDS_WaitSet_wait`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_WaitSet_wait
        (DDS_WaitSet _this,
         DDS_ConditionSeq *active_conditions,
         const DDS_Duration_t *timeout)
```

Description

This operation allows an application thread to wait for the occurrence of at least one of the conditions that is attached to the `DDS_WaitSet`.

Parameters

in `DDS_WaitSet _this` - the `DDS_WaitSet` object on which the operation is operated.

inout `DDS_ConditionSeq *active_conditions` - a sequence which is used to pass the list of all the attached conditions that have a `trigger_value` of `TRUE`.

in *const* `DDS_Duration_t *timeout` - the maximum duration to block for the `DDS_WaitSet_wait`, after which the application thread is unblocked. The special constant `DDS_DURATION_INFINITE` can be used when the maximum waiting time does not need to be bounded.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_TIMEOUT or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation allows an application thread to wait for the occurrence of at least one of the conditions to evaluate to TRUE that is attached to the *DDS_WaitSet*. If all of the conditions attached to the *DDS_WaitSet* have a *trigger_value* of FALSE, the *DDS_WaitSet_wait* operation will block the calling thread. The result of the operation is the continuation of the application thread after which the result is left in *active_conditions*. This is a sequence, which will contain the list of all the attached conditions that have a *trigger_value* of TRUE. The *active_conditions* sequence and its buffer may be pre-allocated by the application and therefore must either be re-used in a subsequent invocation of the *DDS_WaitSet_wait* operation or be released by calling *DDS_free* on the returned *active_conditions*. If the pre-allocated sequence is not big enough to hold the number of triggered *DDS_Conditions*, the sequence will automatically be (re-)allocated to fit the required size. The parameter *timeout* specifies the maximum duration for the *DDS_WaitSet_wait* to block the calling application thread (when none of the attached conditions has a *trigger_value* of TRUE). In that case the return value is *DDS_RETCODE_TIMEOUT* and the *active_conditions* sequence is left empty. Since it is not allowed for more than one application thread to be waiting on the same *DDS_WaitSet*, the operation returns immediately with the value *DDS_RETCODE_PRECONDITION_NOT_MET* when the *DDS_WaitSet_wait* operation is invoked on a *DDS_WaitSet* which already has an application thread blocking on it.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - at least one of the attached conditions has a *trigger_value* of TRUE.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_TIMEOUT* - the timeout has elapsed without any of the attached conditions becoming TRUE.

- *DDS_RETCODE_PRECONDITION_NOT_MET* - the DDS_WaitSet already has an application thread blocking on it.

3.1.7 Class DDS_Condition

This class is the base class for all the conditions that may be attached to a DDS_WaitSet. This base class is specialized in three classes by the Data Distribution Service: DDS_GuardCondition, DDS_StatusCondition and DDS_ReadCondition (also there is a DDS_QueryCondition which is a specialized DDS_ReadCondition).

Each DDS_Condition has a `trigger_value` that can be TRUE or FALSE and is set by the Data Distribution Service (except a DDS_GuardCondition) depending on the evaluation of the DDS_Condition.

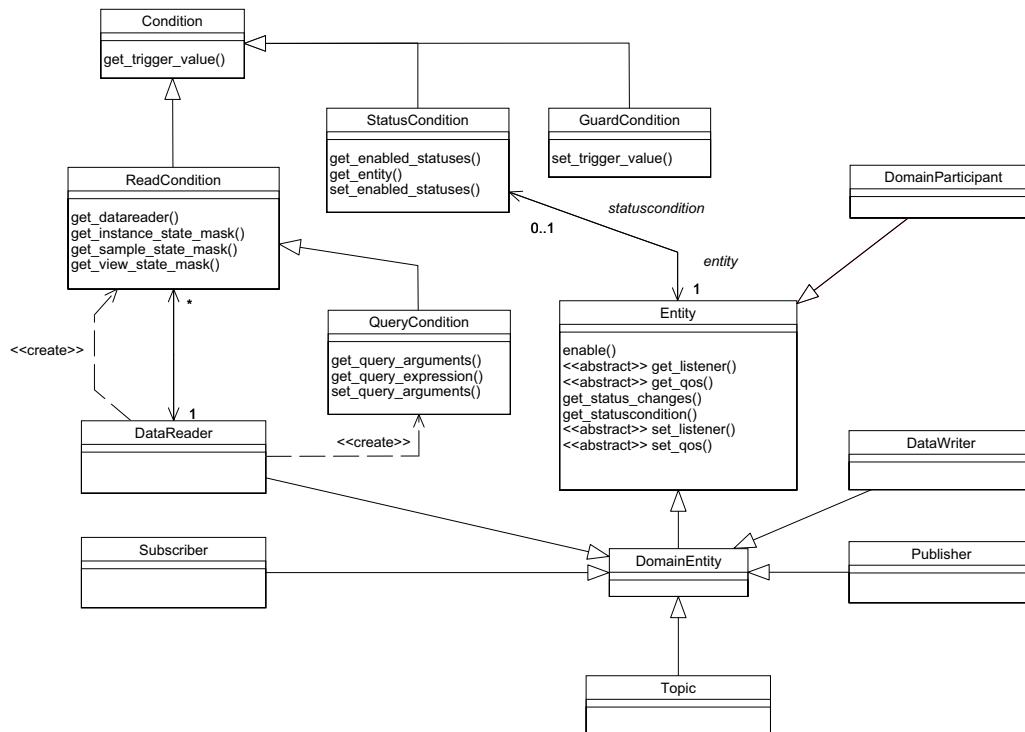


Figure 14 DCPS DDS_Conditions

The interface description of this class is as follows:

/ *


```

* interface DDS_Condition
*/
/*
* implemented API operations
*/
DDS_boolean
DDS_Condition_get_trigger_value
(DDS_Condition _this);

```

The next paragraph describes the usage of the DDS_Condition operation.

3.1.7.1 DDS_Condition_get_trigger_value

Synopsis

```

#include <dds_dcps.h>
DDS_boolean
DDS_Condition_get_trigger_value
(DDS_Condition _this);

```

Description

This operation returns the `trigger_value` of the DDS_Condition.

Parameters

in DDS_Condition _this - the DDS_Condition object on which the operation is operated.

Return Value

DDS_boolean - the trigger_value.

Detailed Description

A DDS_Condition has a `trigger_value` that can be TRUE or FALSE and is set by the Data Distribution Service (except a DDS_GuardCondition). This operation returns the `trigger_value` of the DDS_Condition.

3.1.8 Class DDS_GuardCondition

A DDS_GuardCondition object is a specific DDS_Condition whose `trigger_value` is completely under the control of the application. The DDS_GuardCondition has no factory and must be created by the application. The DDS_GuardCondition is directly created as an object by using the DDS_GuardCondition constructor. When a DDS_GuardCondition is initially created, the `trigger_value` is FALSE. The purpose of the DDS_GuardCondition is to provide the means for an application to manually

wake up a `DDS_WaitSet`. This is accomplished by attaching the `DDS_GuardCondition` to the `Waitset` and setting the `trigger_value` by means of the `DDS_GuardCondition_set_trigger_value` operation.

The interface description of this class is as follows:

```
/*
 * interface DDS_GuardCondition
 */
/*
 * inherited from DDS_Condition
 */
/* DDS_boolean
 *   DDS_GuardCondition_get_trigger_value
 *   (DDS_GuardCondition _this);
 */
/*
 * implemented API operations
 */
DDS_GuardCondition
DDS_GuardCondition__alloc
(void);
DDS_ReturnCode_t
DDS_GuardCondition_set_trigger_value
(DDS_GuardCondition _this,
 const DDS_boolean value);
```

The following sections describe the usage of all `DDS_GuardCondition` operations.

The inherited operation is listed but not fully described since it is not implemented in this class. The full description of this operation is given in the class from which it is inherited. This is described in their respective paragraph.

3.1.8.1 DDS_GuardCondition__alloc

Synopsis

```
#include <dds_dcps.h>
DDS_GuardCondition
DDS_GuardCondition__alloc
(void);
```

Description

This operation creates a new `DDS_GuardCondition`.

Parameters

<none>

Return Value

DDS_GuardCondition - Return value is the handle to the newly created *DDS_GuardCondition*. In case of an error, a *DDS_OBJECT_NIL* pointer is returned.

Detailed Description

This operation creates a new *DDS_GuardCondition*. The *DDS_GuardCondition* must be created using this operation. In other words, the application is not allowed to declare an object of type *DDS_GuardCondition*. When the application wants to release the *DDS_GuardCondition* it must be released using *DDS_free*.

In case there are insufficient resources available to allocate the *DDS_GuardCondition*, a *DDS_OBJECT_NIL* pointer is returned instead.

3.1.8.2 *DDS_GuardCondition_get_trigger_value* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Condition* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_boolean
    DDS_GuardCondition_get_trigger_value
        (DDS_GuardCondition _this);
```

3.1.8.3 *DDS_GuardCondition_set_trigger_value*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_GuardCondition_set_trigger_value
        (DDS_GuardCondition _this,
         const DDS_boolean value);
```

Description

This operation sets the *trigger_value* of the *DDS_GuardCondition*.

Parameters

in DDS_GuardCondition _this - the *DDS_GuardCondition* object on which the operation is operated.

in const DDS_boolean value - the boolean value to which the *DDS_GuardCondition* is set.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR or DDS_RETCODE_ILLEGAL_OPERATION.

Detailed Description

A *DDS_GuardCondition* object is a specific *DDS_Condition* which *trigger_value* is completely under the control of the application. This operation must be used by the application to manually wake-up a *DDS_WaitSet*. This operation sets the *trigger_value* of the *DDS_GuardCondition* to the parameter value. The *DDS_GuardCondition* is directly created using the *DDS_GuardCondition* constructor. When a *DDS_GuardCondition* is initially created, the *trigger_value* is FALSE.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the specified *trigger_value* has successfully been applied
- *DDS_RETCODE_ERROR* - an internal error has occurred
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object

3.1.9 Class *DDS_StatusCondition*

DDS_Entity objects that have status attributes also have a *DDS_StatusCondition*, access is provided to the application by the *DDS_<Entity>_get_statuscondition* operation.

The communication statuses whose changes can be communicated to the application depend on the *DDS_Entity*. The following table shows the relevant statuses for each *DDS_Entity*.

Table 17 Status Per *DDS_Entity*

DDS_Entity	Status Name
DDS_Topic	DDS_INCONSISTENT_TOPIC_STATUS
DDS_Subscriber	DDS_DATA_ON_READERS_STATUS

Table 17 Status Per DDS_Entity

DDS_Entity	Status Name
DDS_DataReader	DDS_SAMPLE_REJECTED_STATUS
	DDS_LIVELINESS_CHANGED_STATUS
	DDS_REQUESTED_DEADLINE_MISSED_STATUS
	DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
	DDS_DATA_AVAILABLE_STATUS
	DDS_SAMPLE_LOST_STATUS
	DDS_SUBSCRIPTION_MATCHED_STATUS
DDS_DataWriter	DDS_LIVELINESS_LOST_STATUS
	DDS_OFFERED_DEADLINE_MISSED_STATUS
	DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
	DDS_PUBLICATION_MATCHED_STATUS

The `trigger_value` of the `DDS_StatusCondition` depends on the communication statuses of that `DDS_Entity` (e.g., missed deadline) and also depends on the value of the `DDS_StatusCondition` attribute mask (enabled_statuses mask). A `DDS_StatusCondition` can be attached to a `DDS_WaitSet` in order to allow an application to suspend until the `trigger_value` has become `TRUE`.

The `trigger_value` of a `DDS_StatusCondition` will be `TRUE` if one of the enabled `StatusChangedFlags` is set. That is, `trigger_value==FALSE` only if all the values of the `StatusChangedFlags` are `FALSE`.

The sensitivity of the `DDS_StatusCondition` to a particular communication status is controlled by the list of `enabled_statuses` set on the condition by means of the `DDS_StatusCondition_set_enabled_statuses` operation.

When the `enabled_statuses` are not changed by the `DDS_StatusCondition_set_enabled_statuses` operation, all statuses are enabled by default.

The interface description of this class is as follows:

```

/*
 * interface DDS_StatusCondition
 */
/*
 * inherited from DDS_Condition
 */
/* DDS_boolean
 *   DDS_StatusCondition_get_trigger_value
 *   (DDS_StatusCondition _this);
 */

```

```

/*
 * implemented API operations
 */
    DDS_StatusMask
        DDS_StatusCondition_get_enabled_statuses
            (DDS_StatusCondition _this);
    DDS_ReturnCode_t
        DDS_StatusCondition_set_enabled_statuses
            (DDS_StatusCondition _this,
             const DDS_StatusMask mask);
    DDS_Entity
        DDS_StatusCondition_get_entity
            (DDS_StatusCondition _this);

```

The next paragraphs describe the usage of all DDS_StatusCondition operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.1.9.1 DDS_StatusCondition_get_enabled_statuses

Synopsis

```

#include <dds_dcps.h>
DDS_StatusMask
    DDS_StatusCondition_get_enabled_statuses
        (DDS_StatusCondition _this);

```

Description

This operation returns the list of enabled communication statuses of the DDS_StatusCondition.

Parameters

in DDS_StatusCondition _this - the DDS_StatusCondition object on which the operation is operated.

Return Value

DDS_StatusMask - Result is a bit-mask in which each bit shows which status is taken into account for the DDS_StatusCondition.

Detailed Description

The trigger_value of the DDS_StatusCondition depends on the communication status of that DDS_Entity (e.g., missed deadline, loss of information, etc.), ‘filtered’ by the set of enabled_statuses on the DDS_StatusCondition.

This operation returns the list of communication statuses that are taken into account to determine the `trigger_value` of the `DDS_StatusCondition`. This operation returns the statuses that were explicitly set on the last call to `DDS_StatusCondition_set_enabled_statuses` or, if `DDS_StatusCondition_set_enabled_statuses` was never called, the default list.

The result value is a bit-mask in which each bit shows which status is taken into account for the `DDS_StatusCondition`. The relevant bits represents one of the following statuses:

```
DDS_INCONSISTENT_TOPIC_STATUS
DDS_OFFERED_DEADLINE_MISSED_STATUS
DDS_REQUESTED_DEADLINE_MISSED_STATUS
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
DDS_SAMPLE_LOST_STATUS
DDS_SAMPLE_REJECTED_STATUS
DDS_DATA_ON_READERS_STATUS
DDS_DATA_AVAILABLE_STATUS
DDS_LIVELINESS_LOST_STATUS
DDS_LIVELINESS_CHANGED_STATUS
DDS_PUBLICATION_MATCHED_STATUS
DDS_SUBSCRIPTION_MATCHED_STATUS.
```

Each status bit is declared as a constant and can be used in an AND operation to check the status bit against the result of type `DDS_StatusMask`. Not all statuses are relevant to all `DDS_Entity` objects. See the respective `Listener` objects for each `DDS_Entity` for more information.

3.1.9.2 DDS_StatusCondition_get_entity

Synopsis

```
#include <dds_dcps.h>
DDS_Entity
    DDS_StatusCondition_get_entity
        (DDS_StatusCondition _this);
```

Description

This operation returns the `DDS_Entity` associated with the `DDS_StatusCondition` or the `DDS_OBJECT_NIL` pointer.

Parameters

in *DDS_StatusCondition _this* - the `DDS_StatusCondition` object on which the operation is operated.

Return Value

DDS_Entity - a pointer to the *DDS_Entity* associated with the *DDS_StatusCondition* or the *DDS_OBJECT_NIL* pointer.

Detailed Description

This operation returns the *DDS_Entity* associated with the *DDS_StatusCondition*. Note that there is exactly one *DDS_Entity* associated with each *DDS_StatusCondition*. When the *DDS_Entity* was already deleted (there is no associated *DDS_Entity* any more), the *DDS_OBJECT_NIL* pointer is returned.

3.1.9.3 DDS_StatusCondition_get_trigger_value (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Condition* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_boolean
    DDS_StatusCondition_get_trigger_value
        (DDS_StatusCondition _this);
```

3.1.9.4 DDS_StatusCondition_set_enabled_statuses**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_StatusCondition_set_enabled_statuses
        (DDS_StatusCondition _this,
         const DDS_StatusMask mask);
```

Description

This operation sets the list of communication statuses that are taken into account to determine the *trigger_value* of the *DDS_StatusCondition*.

Parameters

in DDS_StatusCondition _this - the *DDS_StatusCondition* object on which the operation is operated.

in const DDS_StatusMask mask - a bit-mask in which each bit sets the status which is taken into account for the *DDS_StatusCondition*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION or DDS_RETCODE_ALREADY_DELETED.

Detailed Description

The *trigger_value* of the *DDS_StatusCondition* depends on the communication status of that *DDS_Entity* (e.g., missed deadline, loss of information, etc.), 'filtered' by the set of *enabled_statuses* on the *DDS_StatusCondition*.

This operation sets the list of communication statuses that are taken into account to determine the *trigger_value* of the *DDS_StatusCondition*. This operation may change the *trigger_value* of the *DDS_StatusCondition*.

DDS_WaitSet objects behaviour depend on the changes of the *trigger_value* of their attached *DDS_Conditions*. Therefore, any *DDS_WaitSet* to which the *DDS_StatusCondition* is attached is potentially affected by this operation.

If this function is not invoked, the default list of *enabled_statuses* includes all the statuses.

The parameter *mask* is a bit-mask in which each bit shows which status is taken into account for the *DDS_StatusCondition*. The relevant bits represents one of the following statuses:

```
DDS_INCONSISTENT_TOPIC_STATUS
DDS_OFFERED_DEADLINE_MISSED_STATUS
DDS_REQUESTED_DEADLINE_MISSED_STATUS
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
DDS_SAMPLE_LOST_STATUS
DDS_SAMPLE_REJECTED_STATUS
DDS_DATA_ON_READERS_STATUS
DDS_DATA_AVAILABLE_STATUS
DDS_LIVELINESS_LOST_STATUS
DDS_LIVELINESS_CHANGED_STATUS
DDS_PUBLICATION_MATCHED_STATUS
DDS_SUBSCRIPTION_MATCHED_STATUS
```

Each status bit is declared as a constant and can be used in an OR operation to set the status bit in the parameter *mask* of type *DDS_StatusMask*. Not all statuses are relevant to all *DDS_Entity* objects. See the respective *Listener* objects for each *DDS_Entity* for more information.

Return Code

When the operation returns:

A Domain is a distributed concept that links all the applications that must be able to communicate with each other. It represents a communication plane: only the DDS_Publishers and the DDS_Subscribers attached to the same Domain can interact.

This class implements several functions:

- It acts as a container for all other DDS_Entity objects
- It acts as a factory for the DDS_Publisher, DDS_Subscriber, DDS_Topic, DDS_ContentFilteredTopic and DDS_MultiTopic objects
- It provides access to the built-in DDS_Topic objects
- It provides information about DDS_Topic objects
- It isolates applications within the same Domain (sharing the same domainId) from other applications in a different Domain on the same set of computers. In this way, several independent distributed applications can coexist in the same physical network without interfering, or even being aware of each other
- It provides administration services in the Domain, offering operations, which allow the application to ignore locally any information about a given Participant, Publication, Subscription or Topic.

The interface description of this class is as follows:

```

/*
 * interface DDS_DomainParticipant
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *   DDS_DomainParticipant_get_statuscondition
 *   (DDS_DomainParticipant _this);
 */
/* DDS_StatusMask
 *   DDS_DomainParticipant_get_status_changes
 *   (DDS_DomainParticipant _this);
 */
/* DDS_ReturnCode_t
 *   DDS_DomainParticipant_enable
 *   (DDS_DomainParticipant _this);
 */
/*
 * implemented API operations
 */
DDS_Publisher
DDS_DomainParticipant_create_publisher
(DDS_DomainParticipant _this,
  const DDS_PublisherQos *qos,

```

```

        const struct DDS_PublisherListener *a_listener,
        const DDS_StatusMask mask);
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_publisher
        (DDS_DomainParticipant _this,
        const DDS_Publisher p);
DDS_Subscriber
    DDS_DomainParticipant_create_subscriber
        (DDS_DomainParticipant _this,
        const DDS_SubscriberQos *qos,
        const struct DDS_SubscriberListener *a_listener,
        const DDS_StatusMask mask);
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_subscriber
        (DDS_DomainParticipant _this,
        const DDS_Subscriber s);
DDS_Subscriber
    DDS_DomainParticipant_get_builtin_subscriber
        (DDS_DomainParticipant _this);
DDS_Topic
    DDS_DomainParticipant_create_topic
        (DDS_DomainParticipant _this,
        const DDS_char *topic_name,
        const DDS_char *type_name,
        const DDS_TopicQos *qos,
        const struct DDS_TopicListener *a_listener,
        const DDS_StatusMask mask);
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_topic
        (DDS_DomainParticipant _this,
        const DDS_Topic a_topic);
DDS_Topic
    DDS_DomainParticipant_find_topic
        (DDS_DomainParticipant _this,
        const DDS_char *topic_name,
        const DDS_Duration_t *timeout);
DDS_TopicDescription
    DDS_DomainParticipant_lookup_topicdescription
        (DDS_DomainParticipant _this,
        const DDS_char *name);
DDS_ContentFilteredTopic
    DDS_DomainParticipant_create_contentfilteredtopic
        (DDS_DomainParticipant _this,
        const DDS_char *name,
        const DDS_Topic related_topic,
        const DDS_char *filter_expression,
        const DDS_StringSeq *expression_parameters);
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_contentfilteredtopic
        (DDS_DomainParticipant _this,

```

```

        const DDS_ContentFilteredTopic
            a_contentfilteredtopic);
DDS_MultiTopic
    DDS_DomainParticipant_create_multitopic
        (DDS_DomainParticipant _this,
         const DDS_char *name,
         const DDS_char *type_name,
         const DDS_char *subscription_expression,
         const DDS_StringSeq *expression_parameters);
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_multitopic
        (DDS_DomainParticipant _this,
         const DDS_MultiTopic a_multitopic);
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_contained_entities
        (DDS_DomainParticipant _this);
DDS_ReturnCode_t
    DDS_DomainParticipant_set_qos
        (DDS_DomainParticipant _this,
         const DDS_DomainParticipantQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_qos
        (DDS_DomainParticipant _this,
         DDS_DomainParticipantQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_set_listener
        (DDS_DomainParticipant _this,
         const struct DDS_DomainParticipantListener *a_listener,
         const DDS_StatusMask mask);
struct DDS_DomainParticipantListener
    DDS_DomainParticipant_get_listener
        (DDS_DomainParticipant _this);
DDS_ReturnCode_t
    DDS_DomainParticipant_ignore_participant
        (DDS_DomainParticipant _this,
         const DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
    DDS_DomainParticipant_ignore_topic
        (DDS_DomainParticipant _this,
         const DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
    DDS_DomainParticipant_ignore_publication
        (DDS_DomainParticipant _this,
         const DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
    DDS_DomainParticipant_ignore_subscription
        (DDS_DomainParticipant _this,
         const DDS_InstanceHandle_t handle);
DomainId_t
    DDS_DomainParticipant_get_domain_id

```

```

        (DDS_DomainParticipant _this);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_participants
        (DDS_DomainParticipant _this,
         DDS_InstanceHandleSeq *participant_handles);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_participant_data
        (DDS_DomainParticipant _this,
         DDS_ParticipantBuiltinTopicData *participant_data,
         DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_topics
        (DDS_DomainParticipant _this,
         DDS_InstanceHandleSeq *topic_handles);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_topic_data
        (DDS_DomainParticipant _this,
         DDS_TopicBuiltinTopicData *topic_data,
         DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
    DDS_DomainParticipant_assert_liveliness
        (DDS_DomainParticipant _this);
DDS_ReturnCode_t
    DDS_DomainParticipant_set_default_publisher_qos
        (DDS_DomainParticipant _this,
         const DDS_PublisherQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_default_publisher_qos
        (DDS_DomainParticipant _this,
         DDS_PublisherQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_set_default_subscriber_qos
        (DDS_DomainParticipant _this,
         const DDS_SubscriberQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_default_subscriber_qos
        (DDS_DomainParticipant _this,
         DDS_SubscriberQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_set_default_topic_qos
        (DDS_DomainParticipant _this,
         const DDS_TopicQos *qos);
DDS_ReturnCode_t
    DDS_DomainParticipant_get_default_topic_qos
        (DDS_DomainParticipant _this,
         DDS_TopicQos *qos);
DDS_boolean
    contains_entity
        (DDS_InstanceHandle_t a_handle);
DDS_ReturnCode_t

```

```
get_current_time
(DDS_Time_t *current_time);
```

The following sections describe the usage of all DDS_DomainParticipant operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.2.1.1 DDS_DomainParticipant_assert_liveliness

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_assert_liveliness
        (DDS_DomainParticipant _this);
```

Description

This operation asserts the liveliness for the DDS_DomainParticipant.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_NOT_ENABLED.

Detailed Description

This operation will manually assert the liveliness for the DDS_DomainParticipant. This way, the Data Distribution Service is informed that the DDS_DomainParticipant is still alive. This operation only needs to be used when the DDS_DomainParticipant contains DDS_DataWriters with the DDS_LivelinessQosPolicy set to DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS, and it will only affect the liveliness of those DDS_DataWriters.

Writing data via the DDS_DataWriter_write operation of a DDS_DataWriter will assert the liveliness on the DDS_DataWriter itself and its DDS_DomainParticipant. DDS_DomainParticipant_assert_liveliness subsequently is only needed when data is **not** written regularly.

The liveliness should be asserted by the application, depending on the DDS_LivelinessQosPolicy.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the liveliness of this *DDS_DomainParticipant* has successfully been asserted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DomainParticipant* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *DDS_DomainParticipant* is not enabled.

3.2.1.2 DDS_DomainParticipant_contains_entity**Synopsis**

```
#include <dds_dcps.h>
DDS_boolean
contains_entity
    (DDS_DomainParticipant _this,
     DDS_InstanceHandle_t a_handle);
```

Description

This operation checks whether or not the given Entity represented by *a_handle* is created by the *DDS_DomainParticipant* or any of its contained entities.

Parameters

in DDS_DomainParticipant _this - the *DDS_DomainParticipant* object on which the operation is operated.

in DDS_InstanceHandle_t a_handle - represents a *DDS_Entity* in the Data Distribution System.

Return Value

DDS_boolean - Return value is TRUE if *a_handle* represents a *DDS_Entity* that is created by the *DDS_DomainParticipant* or any of its contained *DDS_Entites*. Otherwise the return value is FALSE.

Detailed Description

This operation checks whether or not the given Entity represented by `a_handle` is created by the `DDS_DomainParticipant` itself (`DDS_TopicDescription`, `DDS_Publisher` or `DDS_Subscriber`) or created by any of its contained entities (`DDS_DataReader`, `DDS_ReadCondition`, `DDS_QueryCondition`, `DDS_DataWriter`, etc.).

Return value is `TRUE` if `a_handle` represents a `DDS_Entity` that is created by the `DDS_DomainParticipant` or any of its contained `DDS_Entites`. Otherwise the return value is `FALSE`.

3.2.1.3 DDS_DomainParticipant_create_contentfilteredtopic

Synopsis

```
#include <dds_dcps.h>
DDS_ContentFilteredTopic
    DDS_DomainParticipant_create_contentfilteredtopic
        (DDS_DomainParticipant _this,
         const DDS_char *name,
         const DDS_Topic related_topic,
         const DDS_char *filter_expression,
         const DDS_StringSeq *expression_parameters);
```

Description

This operation creates a `DDS_ContentFilteredTopic` for a `DDS_DomainParticipant` in order to allow `DDS_DataReaders` to subscribe to a subset of the topic content.

Parameters

in DDS_DomainParticipant _this - the `DDS_DomainParticipant` object on which the operation is operated.

*in const DDS_char *name* - contains the name of the `DDS_ContentFilteredTopic`.

in const DDS_Topic related_topic - the handle to the base `DDS_Topic` on which the filtering will be applied. Therefore, a filtered topic is based on an existing `DDS_Topic`.

*in const DDS_char *filter_expression* - holds the SQL expression (subset of SQL), which defines the filtering.

*in const DDS_StringSeq *expression_parameters* - the handle to a sequence of strings with the parameter value used in the SQL expression (i.e., the number of `%n` tokens in the expression). The number of values in `expression_parameters` must be equal or greater than the highest

referenced %n token in the `filter_expression` (e.g. if %1 and %8 are used as parameter in the `filter_expression`, the `expression_parameters` should at least contain `n+1 = 9` values).

Return Value

`DDS_ContentFilteredTopic` - Return value is the handle to the newly created `DDS_ContentFilteredTopic`. In case of an error, a nil pointer is returned.

Detailed Description

This operation creates a `DDS_ContentFilteredTopic` for a `DDS_DomainParticipant` in order to allow `DDS_DataReaders` to subscribe to a subset of the topic content. The base topic, which is being filtered is defined by the parameter `related_topic`. The resulting `DDS_ContentFilteredTopic` only relates to the samples published under the `related_topic`, which have been filtered according to their content. The resulting `DDS_ContentFilteredTopic` only exists at the `DDS_DataReader` side and will never be published. The samples of the `related_topic` are filtered according to the SQL expression, which is a subset of SQL as defined in the parameter `filter_expression` (see Appendix H, *DCPS Queries and Filters*).

The `filter_expression` may also contain parameters, which appear as %n tokens in the expression which must be set by the sequence of strings defined by the parameter `expression_parameters`. The number of values in `expression_parameters` must be equal or greater than the highest referenced %n token in the `filter_expression` (e.g. if %1 and %8 are used as parameter in the `filter_expression`, the `expression_parameters` should at least contain `n+1 = 9` values).

The `filter_expression` is a string that specifies the criteria to select the data samples of interest. In other words, it identifies the selection of data from the associated `DDS_Topics`. It is an SQL expression where the `WHERE` clause gives the content filter.

3.2.1.4 DDS_DomainParticipant_create_multitopic

Synopsis

```
#include <dds_dcps.h>
DDS_MultiTopic
    DDS_DomainParticipant_create_multitopic
        (DDS_DomainParticipant _this,
         const DDS_char *name,
         const DDS_char *type_name,
         const DDS_char *subscription_expression,
         const DDS_StringSeq *expression_parameters);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

Description

This operation creates a `DDS_MultiTopic` for a `DDS_DomainParticipant` in order to allow `DDS_DataReaders` to subscribe to a filtered/re-arranged combination and/or subset of the content of several topics.

Parameters

in DDS_DomainParticipant _this - the `DDS_DomainParticipant` object on which the operation is operated.

*in const DDS_char *name* - contains the name of the `DDS_MultiTopic`.

*in const DDS_char *type_name* - contains the name of the type of the `DDS_MultiTopic`. This *type_name* must have been registered using `DDS_TypeSupport_register_type` prior to calling this operation.

*in const DDS_char *subscription_expression* - the SQL expression (subset of SQL), which defines the selection, filtering, combining and re-arranging of the sample data.

*in const DDS_StringSeq *expression_parameters* - the handle to a sequence of strings with the parameter value used in the SQL expression (i.e., the number of %n tokens in the expression). The number of values in *expression_parameters* must be equal or greater than the highest referenced %n token in the *subscription_expression* (e.g. if %1 and %8 are used as parameter in the *subscription_expression*, the *expression_parameters* should at least contain $n+1 = 9$ values).

Return Value

DDS_MultiTopic - Return value is the handle to the newly created `DDS_MultiTopic`. In case of an error, a nil pointer is returned.

Detailed Description

This operation creates a `DDS_MultiTopic` for a `DDS_DomainParticipant` in order to allow `DDS_DataReaders` to subscribe to a filtered/re-arranged combination and/or subset of the content of several topics. Before the `DDS_MultiTopic` can be created, the *type_name* of the `DDS_MultiTopic` must have been registered prior to calling this operation. Registering is done, using the `DDS_TypeSupport_register_type` operation from `DDS_TypeSupport`. The list of topics and the logic, which defines the selection, filtering, combining and re-arranging of the sample data, is defined by the SQL expression, a subset of SQL defined in *subscription_expression*. The *subscription_expression* may also contain parameters, which appear as %n tokens in the expression. These

parameters are defined in `expression_parameters`. The number of values in `expression_parameters` must be equal or greater than the highest referenced `%n` token in the `subscription_expression` (e.g. if `%1` and `%8` are used as parameter in the `subscription_expression`, the `expression_parameters` should at least contain `n+1 = 9` values).

The `subscription_expression` is a string that specifies the criteria to select the data samples of interest. In other words, it identifies the selection and rearrangement of data from the associated `DDS_Topics`. It is an SQL expression where the `SELECT` clause provides the fields to be kept, the `FROM` part provides the names of the `DDS_Topics` that are searched for those fields, and the `WHERE` clause gives the content filter. The `DDS_Topics` combined may have different types but they are restricted in that the type of the fields used for the `NATURAL JOIN` operation must be the same.

The `DDS_DataReader`, which is associated with a `DDS_MultiTopic` only accesses information which exist locally in the `DDS_DataReader`, based on the `DDS_Topics` used in the `subscription_expression`. The actual `DDS_MultiTopic` will never be produced, only the individual `DDS_Topics`.

3.2.1.5 DDS_DomainParticipant_create_publisher

Synopsis

```
#include <dds_dcps.h>
DDS_Publisher
    DDS_DomainParticipant_create_publisher
        (DDS_DomainParticipant _this,
         const DDS_PublisherQos *qos,
         const struct DDS_PublisherListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation creates a `DDS_Publisher` with the desired `QosPolicy` settings and if applicable, attaches the optionally specified `DDS_PublisherListener` to it.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

in `const DDS_PublisherQos *qos` - a collection of `QosPolicy` settings for the new `DDS_Publisher`. In case these settings are not self consistent, no `DDS_Publisher` is created.

*in const struct DDS_PublisherListener *a_listener* - a pointer to the DDS_PublisherListener instance which will be attached to the new DDS_Publisher. It is permitted to use DDS_OBJECT_NIL as the value of the listener: this behaves as a DDS_PublisherListener whose operations perform no action.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the DDS_PublisherListener for a certain status.

Return Value

DDS_Publisher - Return value is a pointer to the newly created DDS_Publisher. In case of an error, the DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation creates a DDS_Publisher with the desired QoSPolicy settings and if applicable, attaches the optionally specified DDS_PublisherListener to it. When the DDS_PublisherListener is not applicable, the DDS_OBJECT_NIL pointer must be supplied instead. To delete the DDS_Publisher the operation DDS_DomainParticipant_delete_publisher or DDS_DomainParticipant_delete_contained_entities must be used.

In case the specified QoSPolicy settings are not consistent, no DDS_Publisher is created and the DDS_OBJECT_NIL pointer is returned. DDS_OBJECT_NIL can also be returned when insufficient access rights exist for the partition(s) listed in the provided QoS structure.

Default QoS

The constant DDS_PUBLISHER_QOS_DEFAULT can be used as parameter qos to create a DDS_Publisher with the default DDS_PublisherQos as set in the DDS_DomainParticipant. The effect of using DDS_PUBLISHER_QOS_DEFAULT is the same as calling the operation DDS_DomainParticipant_get_default_publisher_qos and using the resulting DDS_PublisherQos to create the DDS_Publisher.

Communication Status

For each communication status, the StatusChangedFlag flag is initially set to FALSE. It becomes TRUE whenever that communication status changes. For each communication status activated in the mask, the associated DDS_PublisherListener operation is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The fact that the status is reset prior to calling the listener means that if the application calls the get_<status_name>_status from inside the listener it will see the status already reset.

The following statuses are applicable to the `DDS_PublisherListener`:

- `DDS_OFFERED_DEADLINE_MISSED_STATUS` (propagated)
- `DDS_OFFERED_INCOMPATIBLE_QOS_STATUS` (propagated)
- `DDS_LIVELINESS_LOST_STATUS` (propagated)
- `DDS_PUBLICATION_MATCHED_STATUS` (propagated).



Be aware that the `DDS_PUBLICATION_MATCHED_STATUS` is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_OBJECT_NIL`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `DDS_STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

The Data Distribution Service will trigger the most specific and relevant `Listener`. In other words, in case a communication status is also activated on the `DDS_DataWriterListener` of a contained `DDS_DataWriter`, the `DDS_DataWriterListener` on that contained `DDS_DataWriter` is invoked instead of the `DDS_PublisherListener`. This means that a status change on a contained `DDS_DataWriter` only invokes the `DDS_PublisherListener` if the contained `DDS_DataWriter` itself does not handle the trigger event generated by the status change.

In case a communication status is not activated in the mask of the `DDS_PublisherListener`, the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` and a `DDS_Publisher` specific behaviour when needed. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

3.2.1.6 DDS_DomainParticipant_create_subscriber

Synopsis

```
#include <dds_dcps.h>
DDS_Subscriber
    DDS_DomainParticipant_create_subscriber
        (DDS_DomainParticipant _this,
         const DDS_SubscriberQos *qos,
         const struct DDS_SubscriberListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation creates a `DDS_Subscriber` with the desired `QosPolicy` settings and if applicable, attaches the optionally specified `DDS_SubscriberListener` to it.

Parameters

- in* `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.
- in* `const DDS_SubscriberQos *qos` - a collection of `QosPolicy` settings for the new `DDS_Subscriber`. In case these settings are not self consistent, no `DDS_Subscriber` is created.
- in* `const struct DDS_SubscriberListener *a_listener` - a pointer to the `DDS_SubscriberListener` instance which will be attached to the new `DDS_Subscriber`. It is permitted to use `DDS_OBJECT_NIL` as the value of the listener: this behaves as a `DDS_SubscriberListener` whose operations perform no action.
- in* `const DDS_StatusMask mask` - a bit-mask in which each bit enables the invocation of the `DDS_SubscriberListener` for a certain status.

Return Value

`DDS_Subscriber` - Return value is a pointer to the newly created `DDS_Subscriber`. In case of an error, the `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a `DDS_Subscriber` with the desired `QosPolicy` settings and if applicable, attaches the optionally specified `DDS_SubscriberListener` to it. When the `DDS_SubscriberListener` is not applicable, the `DDS_OBJECT_NIL` pointer must be supplied instead. To delete the `DDS_Subscriber` the operation `DDS_DomainParticipant_delete_subscriber` or `DDS_DomainParticipant_delete_contained_entities` must be used.

In case the specified `QoSPolicy` settings are not consistent, no `DDS_Subscriber` is created and the `DDS_OBJECT_NIL` pointer is returned. `DDS_OBJECT_NIL` can also be returned when insufficient access rights exist for the partition(s) listed in the provided `QoS` structure.

Default QoS

The constant `DDS_SUBSCRIBER_QOS_DEFAULT` can be used as parameter `qos` to create a `DDS_Subscriber` with the default `DDS_SubscriberQos` as set in the `DomainParticipant`. The effect of using `DDS_SUBSCRIBER_QOS_DEFAULT` is the same as calling the operation `DDS_DomainParticipant_get_default_subscriber_qos` and using the resulting `DDS_SubscriberQos` to create the `DDS_Subscriber`.

Communication Status

For each communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever that communication status changes. For each communication status activated in the mask, the associated `DDS_SubscriberListener` operation is invoked and the communication status is reset to `FALSE`, as the listener implicitly accesses the status which is passed as a parameter to that operation. The fact that the status is reset prior to calling the listener means that if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset.

The following statuses are applicable to the `DDS_SubscriberListener`:

- `DDS_REQUESTED_DEADLINE_MISSED_STATUS` (propagated)
- `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` (propagated)
- `DDS_SAMPLE_LOST_STATUS` (propagated)
- `DDS_SAMPLE_REJECTED_STATUS` (propagated)
- `DDS_DATA_AVAILABLE_STATUS` (propagated)
- `DDS_LIVELINESS_CHANGED_STATUS` (propagated)
- `DDS_SUBSCRIPTION_MATCHED_STATUS` (propagated)
- `DDS_DATA_ON_READERS_STATUS`



Be aware that the `DDS_SUBSCRIPTION_MATCHED_STATUS` is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_OBJECT_NIL`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `DDS_STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

The Data Distribution Service will trigger the most specific and relevant `Listener`. In other words, in case a communication status is also activated on the `DDS_DataReaderListener` of a contained `DDS_DataReader`, the `DDS_DataReaderListener` on that contained `DDS_DataReader` is invoked instead of the `DDS_SubscriberListener`. This means that a status change on a contained `DDS_DataReader` only invokes the `DDS_SubscriberListener` if the contained `DDS_DataReader` itself does not handle the trigger event generated by the status change.

In case a communication status is not activated in the mask of the `DDS_SubscriberListener`, the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` and a `DDS_Subscriber` specific behaviour when needed. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

The statuses `DDS_DATA_ON_READERS_STATUS` and `DDS_DATA_AVAILABLE_STATUS` are “Read Communication Statuses” and are an exception to all other plain communication statuses: they have no corresponding status structure that can be obtained with a `get_<status_name>_status` operation and they are mutually exclusive. When new information becomes available to a `DataReader`, the Data Distribution Service will first look in an attached and activated `DDS_SubscriberListener` or `DDS_DomainParticipantListener` (in that order) for the `DDS_DATA_ON_READERS_STATUS`. In case the `DDS_DATA_ON_READERS_STATUS` can not be handled, the Data Distribution Service will look in an attached and activated `DDS_DataReaderListener`, `DDS_SubscriberListener` or `DDS_DomainParticipantListener` for the `DDS_DATA_AVAILABLE_STATUS` (in that order).

3.2.1.7 DDS_DomainParticipant_create_topic

Synopsis

```
#include <dds_dcps.h>
DDS_Topic
    DDS_DomainParticipant_create_topic
        (DDS_DomainParticipant _this,
         const DDS_char *topic_name,
         const DDS_char *type_name,
         const DDS_TopicQos *qos,
         const struct DDS_TopicListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation creates a pointer to a new or existing `DDS_Topic` under the given name, for a specific type, with the desired `QosPolicy` settings and if applicable, attaches the optionally specified `DDS_TopicListener` to it.

Parameters

in DDS_DomainParticipant _this - the `DDS_DomainParticipant` object on which the operation is operated.

*in const DDS_char *topic_name* - the name of the `DDS_Topic` to be created. A new `DDS_Topic` will only be created, when no `DDS_Topic`, with the same name, is found within the `DDS_DomainParticipant`.

*in const DDS_char *type_name* - a local alias of the data type, which must have been registered before creating the `DDS_Topic`.

*in const DDS_TopicQos *qos* - a collection of `QosPolicy` settings for the new `DDS_Topic`. In case these settings are not self consistent, no `DDS_Topic` is created.

*in const struct DDS_TopicListener *a_listener* - a pointer to the `DDS_TopicListener` instance which will be attached to the new `DDS_Topic`. It is permitted to use `DDS_OBJECT_NIL` as the value of the listener: this behaves as a `DDS_TopicListener` whose operations perform no action.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the `DDS_TopicListener` for a certain status.

Return Value

DDS_Topic - Return value is a pointer to the new or existing `DDS_Topic`. In case of an error, the `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a pointer to a new or existing `DDS_Topic` under the given name, for a specific type, with the desired `QosPolicy` settings and if applicable, attaches the optionally specified `DDS_TopicListener` to it. When the `DDS_TopicListener` is not applicable, the `DDS_OBJECT_NIL` pointer must be supplied instead. In case the specified `QosPolicy` settings are not consistent, no `DDS_Topic` is created and the `DDS_OBJECT_NIL` pointer is returned. To delete the `DDS_Topic` the operation `DDS_DomainParticipant_delete_topic` or `DDS_DomainParticipant_delete_contained_entities` must be used.

Default QoS

The constant `DDS_TOPIC_QOS_DEFAULT` can be used as parameter `qos` to create a `DDS_Topic` with the default `DDS_TopicQos` as set in the `DDS_DomainParticipant`. The effect of using `DDS_TOPIC_QOS_DEFAULT` is the same as calling the operation `DDS_DomainParticipant_get_default_topic_qos` and using the resulting `DDS_TopicQos` to create the `DDS_Topic`.

The `DDS_Topic` is bound to the type `type_name`. Prior to creating the `DDS_Topic`, the `type_name` must have been registered with the Data Distribution Service. Registering the `type_name` is done using the data type specific `DDS_TypeSupport_register_type` operation.

Existing DDS Topic name

Before creating a new `DDS_Topic`, this operation performs a `DDS_DomainParticipant_lookup_topicdescription` for the specified `topic_name`. When a `DDS_Topic` is found with the same name in the current domain, the `QoS` and `type_name` of the found `DDS_Topic` are matched against the parameters `qos` and `type_name`. When they are the same, no `DDS_Topic` is created but a new proxy of the existing `DDS_Topic` is returned. When they are not exactly the same, no `DDS_Topic` is created and the `DDS_OBJECT_NIL` pointer is returned.

When a `DDS_Topic` is obtained multiple times, it must also be deleted that same number of times using `DDS_DomainParticipant_delete_topic` or calling `DDS_DomainParticipant_delete_contained_entities` once to delete all the proxies.

Local proxy

Since a `DDS_Topic` is a global concept in the system, access is provided through a local proxy. In other words, the pointer returned is actually not a pointer to a `DDS_Topic` but to a locally created proxy. The Data Distribution Service propagates `DDS_Topics` and makes remotely created `DDS_Topics` locally

available through this proxy. For each create, a new proxy is created. Therefore the `DDS_Topic` must be deleted the same number of times, as the `DDS_Topic` was created with the same `topic_name` per Domain. In other words, each pointer (local proxy) must be deleted separately.

Communication Status

For each communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever that communication status changes. For each communication status activated in the `mask`, the associated `DDS_TopicListener` operation is invoked and the communication status is reset to `FALSE`, as the listener implicitly accesses the status which is passed as a parameter to that operation. The fact that the status is reset prior to calling the listener means that if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset.

The following statuses are applicable to the `DDS_TopicListener`:

- `DDS_INCONSISTENT_TOPIC_STATUS`

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `DDS_STATUS_MASK_ANY_V1_2` can be used to select all statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

In case a communication status is not activated in the `mask` of the `DDS_TopicListener`, the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` and a `DDS_Topic` specific behaviour when needed. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its `mask`, the application is not notified of the change.

3.2.1.8 DDS_DomainParticipant_delete_contained_entities

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_contained_entities
        (DDS_DomainParticipant _this);
```

Description

This operation deletes all of the `DDS_Entity` objects that were created on the `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is performed.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation deletes all the `DDS_Entity` objects that were created on the `DDS_DomainParticipant`. In other words, it deletes all `DDS_Publisher`, `DDS_Subscriber`, `DDS_Topic`, `DDS_ContentFilteredTopic` and `DDS_MultiTopic` objects. Prior to deleting each contained `DDS_Entity`, this operation regressively calls the corresponding `DDS_<Entity>_delete_contained_entities` operation on each `DDS_Entity` (if applicable). In other words, all `DDS_Entity` objects in the `DDS_Publisher` and `DDS_Subscriber` are deleted, including the `DDS_DataWriter` and `DDS_DataReader`. Also the `DDS_QueryCondition` and `DDS_ReadCondition` objects contained by the `DDS_DataReader` are deleted.

DDS_Topic

Since a `DDS_Topic` is a global concept in the system, access is provided through a local proxy. The Data Distribution Service propagates `DDS_Topics` and makes remotely created `DDS_Topics` locally available through this proxy. Such a proxy is created by the `DDS_DomainParticipant_create_topic` or `DDS_DomainParticipant_find_topic` operation. When a pointer to the same `DDS_Topic` was created multiple times (either by `DDS_DomainParticipant_create_topic` or `DDS_DomainParticipant_find_topic`), all pointers (local proxies) are deleted. With the last proxy, the `DDS_Topic` itself is also removed from the system.



NOTE: The operation will return `DDS_PRECONDITION_NOT_MET` if the any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained `DDS_DataReader` cannot be deleted because the application has called a `read` or `take` operation and has not called the

corresponding `return_loan` operation to return the loaned samples. In such cases, the operation does not roll back any entity deletions performed prior to the detection of the problem.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the contained `DDS_Entity` objects are deleted and the application may delete the `DDS_DomainParticipant`.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the `DDS_DomainParticipant` has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one or more of the contained entities are in a state where they cannot be deleted.

3.2.1.9 `DDS_DomainParticipant_delete_contentfilteredtopic`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_contentfilteredtopic
        (DDS_DomainParticipant _this,
         const DDS_ContentFilteredTopic
             a_contentfilteredtopic);
```

Description

This operation deletes a `DDS_ContentFilteredTopic`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

in `const DDS_ContentFilteredTopic a_contentfilteredtopic` - a pointer to the `DDS_ContentFilteredTopic`, which is to be deleted.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation deletes a `DDS_ContentFilteredTopic`.

The deletion of a `DDS_ContentFilteredTopic` is not allowed if there are any existing `DDS_DataReader` objects that are using the `DDS_ContentFilteredTopic`.

If the `DDS_DomainParticipant_delete_contentfilteredtopic` operation is called on a `DDS_ContentFilteredTopic` with existing `DDS_DataReader` objects attached to it, it will return `PRECONDITION_NOT_MET`.

The `DDS_DomainParticipant_delete_contentfilteredtopic` operation must be called on the same `DDS_DomainParticipant` object used to create the `DDS_ContentFilteredTopic`.

If `DDS_DomainParticipant_delete_contentfilteredtopic` is called on a different `DDS_DomainParticipant` the operation will have no effect and it will return `PRECONDITION_NOT_MET`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_ContentFilteredTopic` is deleted.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `a_contentfilteredtopic` is not a valid `DDS_ContentFilteredTopic`.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the operation is called on a different `DDS_DomainParticipant`, as used when the `DDS_ContentFilteredTopic` was created, or the `DDS_ContentFilteredTopic` is being used by one or more `DDS_DataReader` objects.

3.2.1.10 DDS_DomainParticipant_delete_multitopic

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_multitopic
        (DDS_DomainParticipant _this,
         const DDS_MultiTopic a_multitopic);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

Description

This operation deletes a DDS_MultiTopic.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

in const DDS_MultiTopic a_multitopic - a pointer to the DDS_MultiTopic, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation deletes a DDS_MultiTopic.

The deletion of a DDS_MultiTopic is not allowed if there are any existing DDS_DataReader objects that are using the DDS_MultiTopic. If the DDS_DomainParticipant_delete_multitopic operation is called on a DDS_MultiTopic with existing DDS_DataReader objects attached to it, it will return DDS_RETCODE_PRECONDITION_NOT_MET.

The DDS_DomainParticipant_delete_multitopic operation must be called on the same DDS_DomainParticipant object used to create the DDS_MultiTopic. If DDS_DomainParticipant_delete_multitopic is called on a different DDS_DomainParticipant the operation will have no effect and it will return DDS_RETCODE_PRECONDITION_NOT_MET.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the *DDS_MultiTopic* is deleted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *a_multitopic* is not a valid *DDS_MultiTopic*.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DomainParticipant* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the operation is called on a different *DDS_DomainParticipant*, as used when the *DDS_MultiTopic* was created, or the *DDS_MultiTopic* is being used by one or more *DDS_DataReader* objects.

3.2.1.II *DDS_DomainParticipant_delete_publisher*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_publisher
        (DDS_DomainParticipant _this,
         const DDS_Publisher p);
```

Description

This operation deletes a *DDS_Publisher*.

Parameters

in DDS_DomainParticipant _this - the *DDS_DomainParticipant* object on which the operation is operated.

in const DDS_Publisher p - a pointer to the *DDS_Publisher*, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_PRECONDITION_NOT_MET*.

Detailed Description

This operation deletes a `DDS_Publisher`. A `DDS_Publisher` cannot be deleted when it has any attached `DDS_DataWriter` objects. When the operation is called on a `DDS_Publisher` with `DDS_DataWriter` objects, the operation returns `DDS_RETCODE_PRECONDITION_NOT_MET`. When the operation is called on a different `DDS_DomainParticipant`, as used when the `DDS_Publisher` was created, the operation has no effect and returns `DDS_RETCODE_PRECONDITION_NOT_MET`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_Publisher` is deleted.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `p` is not a valid `DDS_Publisher`.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the operation is called on a different `DDS_DomainParticipant`, as used when the `DDS_Publisher` was created, or the `DDS_Publisher` contains one or more `DDS_DataWriter` objects.

3.2.1.12 `DDS_DomainParticipant_delete_subscriber`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_delete_subscriber
        (DDS_DomainParticipant _this,
         const DDS_Subscriber s);
```

Description

This operation deletes a `DDS_Subscriber`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

in *const DDS_Subscriber s* - a pointer to the DDS_Subscriber, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation deletes a DDS_Subscriber. A DDS_Subscriber cannot be deleted when it has any attached DDS_DataReader objects. When the operation is called on a DDS_Subscriber with DDS_DataReader objects, the operation returns DDS_RETCODE_PRECONDITION_NOT_MET. When the operation is called on a different DDS_DomainParticipant, as used when the DDS_Subscriber was created, the operation has no effect and returns DDS_RETCODE_PRECONDITION_NOT_MET.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_Subscriber is deleted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *s* is not a valid DDS_Subscriber.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_DomainParticipant has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the operation is called on a different DDS_DomainParticipant, as used when the DDS_Subscriber was created, or the DDS_Subscriber contains one or more DDS_DataReader objects.

3.2.1.13 DDS_DomainParticipant_delete_topic

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
```

```
DDS_DomainParticipant_delete_topic
(DDS_DomainParticipant _this,
 const DDS_Topic a_topic);
```

Description

This operation deletes a DDS_Topic.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

in const DDS_Topic a_topic - a pointer to the DDS_Topic, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation deletes a DDS_Topic. A DDS_Topic cannot be deleted when there are any DDS_DataReader, DDS_DataWriter, DDS_ContentFilteredTopic or DDS_MultiTopic objects, which are using the DDS_Topic. When the operation is called on a DDS_Topic pointed to by any of these objects, the operation returns DDS_RETCODE_PRECONDITION_NOT_MET. When the operation is called on a different DDS_DomainParticipant, as used when the DDS_Topic was created, the operation has no effect and returns DDS_RETCODE_PRECONDITION_NOT_MET.

Local Proxy

Since a DDS_Topic is a global concept in the system, access is provided through a local proxy. In other words, the pointer is actually not a pointer to a DDS_Topic but to the local proxy. The Data Distribution Service propagates DDS_Topics and makes remotely created DDS_Topics locally available through this proxy. Such a proxy is created by the DDS_DomainParticipant_create_topic or DDS_DomainParticipant_find_topic operation. This operation will delete the local proxy. When a pointer to the same DDS_Topic was created multiple times (either by DDS_DomainParticipant_create_topic or DDS_DomainParticipant_find_topic), each pointer (local proxy) must be deleted separately. When this proxy is the last proxy for this DDS_Topic, the

DDS_Topic itself is also removed from the system. As mentioned, a proxy may only be deleted when there are no other entities attached to it. However, it is possible to delete a proxy while there are entities attached to a different proxy.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_Topic is deleted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *a_topic* is not a valid DDS_Topic.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_DomainParticipant has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the operation is called on a different DDS_DomainParticipant, as used when the DDS_Topic was created, or the DDS_Topic is still pointed to by other objects.

3.2.1.14 DDS_DomainParticipant_enable (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_enable
        (DDS_DomainParticipant _this);
```

3.2.1.15 DDS_DomainParticipant_find_topic

Synopsis

```
#include <dds_dcps.h>
DDS_Topic
    DDS_DomainParticipant_find_topic
        (DDS_DomainParticipant _this,
         const DDS_char *topic_name,
         const DDS_Duration_t *timeout);
```

Description

This operation gives access to an existing (or ready to exist) enabled `DDS_Topic`, based on its `topic_name`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

in `const DDS_char *topic_name` - the name of the `DDS_Topic` that the application wants access to.

in `const DDS_Duration_t *timeout` - the maximum duration to block for the `DDS_DomainParticipant_find_topic`, after which the application thread is unblocked. The special constant `DDS_DURATION_INFINITE` can be used when the maximum waiting time does not need to be bounded.

Return Value

`DDS_Topic` - Return value is a pointer to the `DDS_Topic` found.

Detailed Description

This operation gives access to an existing `DDS_Topic`, based on its `topic_name`. The operation takes as arguments the `topic_name` of the `DDS_Topic` and a `timeout`.

If a `DDS_Topic` of the same `topic_name` already exists, it gives access to this `DDS_Topic`. Otherwise it waits (blocks the caller) until another mechanism creates it. This other mechanism can be another thread, a configuration tool, or some other Data Distribution Service utility. If after the specified `timeout` the `DDS_Topic` can still not be found, the caller gets unblocked and `DDS_HANDLE_NIL` is returned.

A `DDS_Topic` obtained by means of `DDS_DomainParticipant_find_topic`, must also be deleted by means of `DDS_DomainParticipant_delete_topic` so that the local resources can be released. If a `DDS_Topic` is obtained multiple times it must also be deleted that same number of times using `DDS_DomainParticipant_delete_topic` or calling `DDS_DomainParticipant_delete_contained_entities` once to delete all the proxies.

A `DDS_Topic` that is obtained by means of `DDS_DomainParticipant_find_topic` in a specific `DDS_DomainParticipant` can only be used to create `DDS_DataReaders` and `DDS_DataWriters` in that `DDS_DomainParticipant` if its corresponding `DDS_TypeSupport` has been registered to that same `DDS_DomainParticipant`.

Local Proxy

Since a `DDS_Topic` is a global concept in the system, access is provided through a local proxy. In other words, the pointer returned is actually not a pointer to a `DDS_Topic` but to a locally created proxy. The Data Distribution Service propagates `DDS_Topics` and makes remotely created `DDS_Topics` locally available through this proxy. For each time this operation is called, a new proxy is created. Therefore the `DDS_Topic` must be deleted the same number of times, as the `DDS_Topic` was created with the same `topic_name` per Domain. In other words, each pointer (local proxy) must be deleted separately.

3.2.1.16 `DDS_DomainParticipant_get_builtin_subscriber`

Synopsis

```
#include <dds_dcps.h>
DDS_Subscriber
    DDS_DomainParticipant_get_builtin_subscriber
        (DDS_DomainParticipant _this);
```

Description

This operation returns the built-in `DDS_Subscriber` associated with the `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

Return Value

`DDS_Subscriber` - Result value is a pointer to the built-in `DDS_Subscriber` associated with the `DDS_DomainParticipant`.

Detailed Description

This operation returns the built-in `DDS_Subscriber` associated with the `DDS_DomainParticipant`. Each `DDS_DomainParticipant` contains several built-in `DDS_Topic` objects. The built-in `DDS_Subscriber` contains the corresponding `DDS_DataReader` objects to access them. All these `DDS_DataReader` objects belong to a single built-in `DDS_Subscriber`. Note that there is exactly one built-in `DDS_Subscriber` associated with each `DDS_DomainParticipant`.

3.2.1.17 `DDS_DomainParticipant_get_current_time`

Synopsis

```
#include <dds_dcps.h>
```

```

DDS_ReturnCode_t
    get_current_time
        (DDS_DomainParticipant _this,
         DDS_Time_t *current_time);

```

Description

This operation returns the value of the current time that the Data Distribution Service uses to time-stamp written data as well as received data in `current_time`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

inout `DDS_Time_t *current_time` - the value of the current time as used by the Data Distribution System. The input value of `current_time` is ignored.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES` or `DDS_RETCODE_NOT_ENABLED`.

Detailed Description

This operation returns the value of the current time that the Data Distribution Service uses to time-stamp written data as well as received data in `current_time`. The input value of `current_time` is ignored by the operation.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the value of the current time is returned in `current_time`.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `current_time` is not a valid reference.
- `DDS_RETCODE_ALREADY_DELETED` - the `DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_DomainParticipant` is not enabled.

3.2.1.18 DDS_DomainParticipant_get_default_publisher_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_default_publisher_qos
        (DDS_DomainParticipant _this,
         DDS_PublisherQos *qos);
```

Description

This operation gets the struct with the default DDS_Publisher QoSPolicy settings of the DDS_DomainParticipant.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

*inout DDS_PublisherQos *qos* - a pointer to the DDS_PublisherQos struct (provided by the application) in which the default QoSPolicy settings for the DDS_Publisher are written.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation gets the struct with the default DDS_Publisher QoSPolicy settings of the DDS_DomainParticipant (that is the DDS_PublisherQos) which is used for newly created DDS_Publisher objects, in case the constant DDS_PUBLISHER_QOS_DEFAULT is used. The default DDS_PublisherQos is only used when the constant is supplied as parameter qos to specify the DDS_PublisherQos in the DDS_DomainParticipant_create_publisher operation. The application must provide the DDS_PublisherQos struct in which the QoSPolicy settings can be stored and pass the qos pointer to the operation. The operation writes the default QoSPolicy settings to the struct pointed to by qos. Any settings in the struct are overwritten.

The values retrieved by this operation match the set of values specified on the last successful call to DDS_DomainParticipant_set_default_publisher_qos, or, if the call was never made, the default values as specified for each QoSPolicy setting as defined in Table 5 on page 59.



NOTE: The operation will return `DDS_PRECONDITION_NOT_MET` if the any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained `DDS_DataReader` cannot be deleted because the application has called a `read` or `take` operation and has not called the corresponding `return_loan` operation to return the loaned samples. In such cases, the operation does not roll back any entity deletions performed prior to the detection of the problem.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the default `DDS_Publisher QosPolicy` settings of this `DDS_DomainParticipant` have successfully been copied into the specified `DDS_PublisherQos` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one or more of the contained entities are in a state where they cannot be deleted.

3.2.1.19 `DDS_DomainParticipant_get_default_subscriber_qos`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_default_subscriber_qos
        (DDS_DomainParticipant _this,
         DDS_SubscriberQos *qos);
```

Description

This operation gets the struct with the default `DDS_Subscriber QosPolicy` settings of the `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

*inout DDS_SubscriberQos *qos* - a pointer to the QosPolicy struct (provided by the application) in which the default QosPolicy settings for the DDS_Subscriber is written.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation gets the struct with the default DDS_Subscriber QosPolicy settings of the DDS_DomainParticipant (that is the DDS_SubscriberQos) which is used for newly created DDS_Subscriber objects, in case the constant DDS_SUBSCRIBER_QOS_DEFAULT is used. The default DDS_SubscriberQos is only used when the constant is supplied as parameter qos to specify the DDS_SubscriberQos in the DDS_DomainParticipant_create_subscriber operation. The application must provide the QoS struct in which the policy can be stored and pass the qos pointer to the operation. The operation writes the default QosPolicy to the struct pointed to by qos. Any settings in the struct are overwritten.

The values retrieved by this operation match the set of values specified on the last successful call to DDS_DomainParticipant_set_default_subscriber_qos, or, if the call was never made, the default values as specified for each QosPolicy defined in Table 5 on page 59.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the default DDS_Subscriber QosPolicy settings of this DDS_DomainParticipant have successfully been copied into the specified DDS_SubscriberQos parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_DomainParticipant has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.1.20 DDS_DomainParticipant_get_default_topic_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_default_topic_qos
        (DDS_DomainParticipant _this,
         DDS_TopicQos *qos);
```

Description

This operation gets the struct with the default DDS_Topic QoSPolicy settings of the DDS_DomainParticipant.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

inout DDS_TopicQos *qos - a pointer to the QoSPolicy struct (provided by the application) in which the default QoSPolicy settings for the DDS_Topic is written.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation gets the struct with the default DDS_Topic QoSPolicy settings of the DDS_DomainParticipant (that is the DDS_TopicQos) which is used for newly created DDS_Topic objects, in case the constant DDS_TOPIC_QOS_DEFAULT is used. The default DDS_TopicQos is only used when the constant is supplied as parameter qos to specify the DDS_TopicQos in the DDS_DomainParticipant_create_topic operation. The application must provide the QoS struct in which the policy can be stored and pass the qos pointer to the operation. The operation writes the default QoSPolicy to the struct pointed to by qos. Any settings in the struct are overwritten.

The values retrieved by this operation match the set of values specified on the last successful call to DDS_DomainParticipant_set_default_topic_qos, or, if the call was never made, the default values as specified for each QoSPolicy defined in Table 5 on page 59.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the default `DDS_Topic QosPolicy` settings of this `DDS_DomainParticipant` have successfully been copied into the specified `DDS_TopicQos` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.2.1.21 `DDS_DomainParticipant_get_discovered_participants`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_participants
        (DDS_DomainParticipant _this,
         DDS_InstanceHandleSeq *participant_handles);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.22 `DDS_DomainParticipant_get_discovered_participant_data`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_participant_data
        (DDS_DomainParticipant _this,
         DDS_ParticipantBuiltinTopicData *participant_data,
         DDS_InstanceHandle_t handle);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.23 `DDS_DomainParticipant_get_discovered_topics`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_topics
        (DDS_DomainParticipant _this,
         DDS_InstanceHandleSeq *topic_handles);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.24 DDS_DomainParticipant_get_discovered_topic_data

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_discovered_topic_data
        (DDS_DomainParticipant _this,
         DDS_TopicBuiltinTopicData *topic_data,
         DDS_InstanceHandle_t handle);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.25 DDS_DomainParticipant_get_domain_id

Synopsis

```
#include <dds_dcps.h>
DomainId_t
    DDS_DomainParticipant_get_domain_id
        (DDS_DomainParticipant _this);
```

Description

This operation returns the `DomainId` of the Domain to which this `DDS_DomainParticipant` is attached.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

Return Value

`DomainId_t` - result is the `DomainId`.

Detailed Description

This operation returns the `DomainId` of the Domain to which this `DDS_DomainParticipant` is attached. A `DomainId` consists of a string that represents either a URI to the location of the configuration file (e.g. "file:///projects/DDS/ospl.xml") or the Domain name as specified in the configuration file. The actual value returned is dependent of the value used when creating the `DomainParticipant`, also see the `DDS_DomainParticipantFactory_create_participant` operation. If a `DomainParticipant` is created using the Domain name then it will also return the

Domain name, and *vice versa*: when created using a URI then the URI will be returned by this operation. The configuration file, identified by the URI, specifies all configuration details of the Domain.

A DomainId may contain the NULL pointer: in that case the location of the configuration file is extracted from the environment variable called OSPL_URI.

3.2.1.26 DDS_DomainParticipant_get_listener

Synopsis

```
#include <dds_dcps.h>
struct DDS_DomainParticipantListener
    DDS_DomainParticipant_get_listener
        (DDS_DomainParticipant _this);
```

Description

This operation allows access to a DDS_DomainParticipantListener.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

Return Value

struct DDS_DomainParticipantListener - a pointer to the DDS_DomainParticipantListener attached to the DDS_DomainParticipant.

Detailed Description

This operation allows access to a DDS_DomainParticipantListener attached to the DDS_DomainParticipant. When no DDS_DomainParticipantListener was attached to the DDS_DomainParticipant, the DDS_OBJECT_NIL pointer is returned.

3.2.1.27 DDS_DomainParticipant_get_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_get_qos
        (DDS_DomainParticipant _this,
         DDS_DomainParticipantQos *qos);
```

Description

This operation allows access to the existing set of QoS policies for a `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

inout `DDS_DomainParticipantQos *qos` - a pointer to the destination `DDS_DomainParticipantQos` struct in which the `QosPolicy` settings will be copied.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation allows access to the existing set of QoS policies of a `DDS_DomainParticipant` on which this operation is used. This `DDS_DomainParticipantQos` is stored at the location pointed to by the `qos` parameter.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of QoS policy values applied to this `DDS_DomainParticipant` has successfully been copied into the specified `DDS_DomainParticipantQos` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.2.1.28 `DDS_DomainParticipant_get_status_changes` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
DDS_DomainParticipant_get_status_changes
(DDS_DomainParticipant _this);
```

3.2.1.29 DDS_DomainParticipant_get_statuscondition (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
DDS_DomainParticipant_get_statuscondition
(DDS_DomainParticipant _this);
```

3.2.1.30 DDS_DomainParticipant_ignore_participant**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DomainParticipant_ignore_participant
(DDS_DomainParticipant _this,
 const DDS_InstanceHandle_t handle);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.31 DDS_DomainParticipant_ignore_publication**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DomainParticipant_ignore_publication
(DDS_DomainParticipant _this,
 const DDS_InstanceHandle_t handle);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.32 DDS_DomainParticipant_ignore_subscription**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DomainParticipant_ignore_subscription
(DDS_DomainParticipant _this,
 const DDS_InstanceHandle_t handle);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.33 DDS_DomainParticipant_ignore_topic

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_ignore_topic
        (DDS_DomainParticipant _this,
         const DDS_InstanceHandle_t handle);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.2.1.34 DDS_DomainParticipant_lookup_topicdescription

Synopsis

```
#include <dds_dcps.h>
DDS_TopicDescription
    DDS_DomainParticipant_lookup_topicdescription
        (DDS_DomainParticipant _this,
         const DDS_char *name);
```

Description

This operation gives access to a locally-created DDS_TopicDescription, with a matching name.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

*in const DDS_char *name* - the name of the DDS_TopicDescription to look for.

Return Value

DDS_TopicDescription - Return value is a pointer to the DDS_TopicDescription found. When no such DDS_TopicDescription is found, the DDS_OBJECT_NIL pointer is returned.

Detailed Description

The operation DDS_DomainParticipant_lookup_topicdescription gives access to a locally-created DDS_TopicDescription, based on its name. The operation takes as argument the name of the DDS_TopicDescription.

If one or more local DDS_TopicDescription proxies (also see Section 3.2.1.15, *DDS_DomainParticipant_find_topic*, on page 163) of the same name already exist, a pointer to one of the already existing local proxies is returned: DDS_DomainParticipant_lookup_topicdescription will never create a

new local proxy. That means that the proxy that is returned does not need to be deleted separately from its original. When no local proxy exists, it returns the `DDS_OBJECT_NIL` pointer. The operation never blocks.

The operation `DDS_DomainParticipant_lookup_topicdescription` may be used to locate any locally-created `DDS_Topic`, `DDS_ContentFilteredTopic` and `DDS_MultiTopic` object.

3.2.1.35 `DDS_DomainParticipant_set_default_publisher_qos`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_set_default_publisher_qos
        (DDS_DomainParticipant _this,
         const DDS_PublisherQos *qos);
```

Description

This operation sets the default `DDS_PublisherQos` of the `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.

in `const DDS_PublisherQos *qos` - a collection of `QosPolicy` settings, which contains the new default `QosPolicy` settings for the newly created `DDS_Publishers`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_UNSUPPORTED`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation sets the default `DDS_PublisherQos` of the `DDS_DomainParticipant` (that is the struct with the `QosPolicy` settings) which is used for newly created `DDS_Publisher` objects, in case the constant `DDS_PUBLISHER_QOS_DEFAULT` is used. The default `DDS_PublisherQos` is only used when the constant is supplied as parameter `qos` to specify the `DDS_PublisherQos` in the `DDS_DomainParticipant_create_publisher` operation. The `DDS_PublisherQos` is always self consistent, because its policies

do not depend on each other. This means this operation never returns the `DDS_RETCODE_INCONSISTENT_POLICY`. The values set by this operation are returned by `DDS_DomainParticipant_get_default_publisher_qos`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the new default `DDS_PublisherQos` is set.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `qos` is not a valid `DDS_PublisherQos`. It contains a `QosPolicy` setting with an enum value that is outside its legal boundaries, or a sequence that has inconsistent memory settings.
- `DDS_RETCODE_UNSUPPORTED` - one or more of the selected `QosPolicy` values are currently not supported by OpenSplice.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.2.1.36 `DDS_DomainParticipant_set_default_subscriber_qos`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_set_default_subscriber_qos
        (DDS_DomainParticipant _this,
         const DDS_SubscriberQos *qos);
```

Description

This operation sets the default `DDS_SubscriberQos` of the `DDS_DomainParticipant`.

Parameters

- in* `DDS_DomainParticipant _this` - the `DDS_DomainParticipant` object on which the operation is operated.
- in* `const DDS_SubscriberQos *qos` - a collection of `QosPolicy` settings, which contains the new default `QosPolicy` settings for the newly created `DDS_Subscribers`.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation sets the default *DDS_SubscriberQos* of the *DDS_DomainParticipant* (that is the struct with the *QosPolicy* settings) which is used for newly created *DDS_Subscriber* objects, in case the constant *DDS_SUBSCRIBER_QOS_DEFAULT* is used. The default *DDS_SubscriberQos* is only used when the constant is supplied as parameter *qos* to specify the *DDS_SubscriberQos* in the *DDS_DomainParticipant_create_subscriber* operation. The *DDS_SubscriberQos* is always self consistent, because its policies do not depend on each other. This means this operation never returns the *DDS_RETCODE_INCONSISTENT_POLICY*. The values set by this operation are returned by *DDS_DomainParticipant_get_default_subscriber_qos*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new default *DDS_SubscriberQos* is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *qos* is not a valid *DDS_PublisherQos*. It contains a *QosPolicy* setting with an enum value that is outside its legal boundaries, or a sequence that has inconsistent memory settings.
- *DDS_RETCODE_UNSUPPORTED* - one or more of the selected *QosPolicy* values are currently not supported by OpenSplice.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DomainParticipant* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.1.37 *DDS_DomainParticipant_set_default_topic_qos*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
```

```
DDS_DomainParticipant_set_default_topic_qos
(DDS_DomainParticipant _this,
 const DDS_TopicQos *qos);
```

Description

This operation sets the default DDS_TopicQos of the DDS_DomainParticipant.

Parameters

in DDS_DomainParticipant _this - the DDS_DomainParticipant object on which the operation is operated.

*in const DDS_TopicQos *qos* - a collection of QosPolicy settings, which contains the new default QosPolicy settings for the newly created DDS_Topics.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_INCONSISTENT_POLICY.

Detailed Description

This operation sets the default DDS_TopicQos of the DDS_DomainParticipant (that is the struct with the QosPolicy settings) which is used for newly created DDS_Topic objects, in case the constant DDS_TOPIC_QOS_DEFAULT is used. The default DDS_TopicQos is only used when the constant is supplied as parameter qos to specify the DDS_TopicQos in the DDS_DomainParticipant_create_topic operation. This operation checks if the DDS_TopicQos is self consistent. If it is not, the operation has no effect and returns DDS_RETCODE_INCONSISTENT_POLICY. The values set by this operation are returned by DDS_DomainParticipant_get_default_topic_qos.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new default DDS_TopicQos is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.

- *DDS_RETCODE_BAD_PARAMETER* - the parameter *qos* is not a valid *DDS_TopicQos*. It contains a *QosPolicy* setting with an invalid *DDS_Duration_t* value, an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- *DDS_RETCODE_UNSUPPORTED* - one or more of the selected *QosPolicy* values are currently not supported by OpenSplice.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DomainParticipant* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_INCONSISTENT_POLICY* - the parameter *qos* contains conflicting *QosPolicy* settings, e.g. a history depth that is higher than the specified resource limits.

3.2.1.38 *DDS_DomainParticipant_set_listener*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_set_listener
        (DDS_DomainParticipant _this,
         const struct DDS_DomainParticipantListener
             *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation attaches a *DDS_DomainParticipantListener* to the *DDS_DomainParticipant*.

Parameters

- in DDS_DomainParticipant _this* - the *DDS_DomainParticipant* object on which the operation is operated.
- in const struct DDS_DomainParticipantListener *a_listener* - a pointer to the *DDS_DomainParticipantListener* instance, which will be attached to the *DDS_DomainParticipant*.
- in const DDS_StatusMask mask* - a bit-mask in which each bit enables the invocation of the *DDS_DomainParticipantListener* for a certain status.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_UNSUPPORTED`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation attaches a `DDS_DomainParticipantListener` to the `DDS_DomainParticipant`. Only one `DDS_DomainParticipantListener` can be attached to each `DDS_DomainParticipant`. If a `DDS_DomainParticipantListener` was already attached, the operation will replace it with the new one. When a `_listener` is the `DDS_OBJECT_NIL` pointer, it represents a listener that is treated as a NOOP¹ for all statuses activated in the bitmask.

Communication Status

For each communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever that communication status changes. For each communication status activated in the mask, the associated `DDS_DomainParticipantListener` operation is invoked and the communication status is reset to `FALSE`, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset. An exception to this rule is the `DDS_OBJECT_NIL` listener, which does not reset the communication statuses for which it is invoked.

The following statuses are applicable to the `DDS_DomainParticipantListener`:

- `DDS_INCONSISTENT_TOPIC_STATUS` *(propagated)*
- `DDS_OFFERED_DEADLINE_MISSED_STATUS` *(propagated)*
- `DDS_REQUESTED_DEADLINE_MISSED_STATUS` *(propagated)*
- `DDS_OFFERED_INCOMPATIBLE_QOS_STATUS` *(propagated)*
- `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` *(propagated)*
- `DDS_SAMPLE_LOST_STATUS` *(propagated)*
- `DDS_SAMPLE_REJECTED_STATUS` *(propagated)*
- `DDS_DATA_ON_READERS_STATUS` *(propagated)*
- `DDS_DATA_AVAILABLE_STATUS` *(propagated)*

1. Short for **No-Operation**, an instruction that performs nothing at all.

- `DDS_LIVELINESS_LOST_STATUS` *(propagated)*
- `DDS_LIVELINESS_CHANGED_STATUS` *(propagated)*
- `DDS_PUBLICATION_MATCHED_STATUS` *(propagated)*
- `DDS_SUBSCRIPTION_MATCHED_STATUS` *(propagated).*



Be aware that the `DDS_PUBLICATION_MATCHED_STATUS` and `DDS_SUBSCRIPTION_MATCHED_STATUS` are not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

The Data Distribution Service will trigger the most specific and relevant `Listener`. In other words, in case a communication status is also activated on the `Listener` of a contained entity, the `Listener` on that contained entity is invoked instead of the `DDS_DomainParticipantListener`. This means that a status change on a contained entity only invokes the `DDS_DomainParticipantListener` if the contained entity itself does not handle the trigger event generated by the status change.

The statuses `DDS_DATA_ON_READERS_STATUS` and `DDS_DATA_AVAILABLE_STATUS` are “Read Communication Statuses” and are an exception to all other plain communication statuses: they have no corresponding status structure that can be obtained with a `get_<status_name>_status` operation and they are mutually exclusive. When new information becomes available to a `DataReader`, the Data Distribution Service will first look in an attached and activated `DDS_SubscriberListener` or `DDS_DomainParticipantListener` (in that order) for the `DDS_DATA_ON_READERS_STATUS`. In case the `DDS_DATA_ON_READERS_STATUS` can not be handled, the Data Distribution Service will look in an attached and activated `DDS_DataReaderListener`, `DDS_SubscriberListener` or `DDS_DomainParticipantListener` for the `DDS_DATA_AVAILABLE_STATUS` (in that order).

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the *DDS_DomainParticipantListener* is attached.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_UNSUPPORTED* - a status was selected that cannot be supported because the infrastructure does not maintain the required connectivity information.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DomainParticipant* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.1.39 *DDS_DomainParticipant_set_qos*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipant_set_qos
        (DDS_DomainParticipant _this,
         const DDS_DomainParticipantQos *qos);
```

Description

This operation replaces the existing set of *QosPolicy* settings for a *DDS_DomainParticipant*.

Parameters

in DDS_DomainParticipant _this - is the *DDS_DomainParticipant* object on which the operation is operated.

*in const DDS_DomainParticipantQos *qos* - must contain the new set of *QosPolicy* settings for the *DDS_DomainParticipant*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation replaces the existing set of `QosPolicy` settings for a `DDS_DomainParticipant`. The parameter `qos` must contain the struct with the `QosPolicy` settings which is checked for self-consistency.

The set of `QosPolicy` settings specified by the `qos` parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned `DDS_RETCODE_OK`).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the new `DDS_DomainParticipantQos` is set.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `qos` is not a valid `DDS_DomainParticipantQos`. It contains a `QosPolicy` setting with a sequence that has inconsistent memory settings.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DomainParticipant` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.2.2 Class `DDS_DomainParticipantFactory`

The purpose of this class is to allow the creation and destruction of `DDS_DomainParticipant` objects. `DDS_DomainParticipantFactory` itself has no factory. It is a pre-existing singleton object that can be accessed by means of the `DDS_DomainParticipantFactory_get_instance` operation on the `DDS_DomainParticipantFactory` class.

The pre-defined value `DDS_TheParticipantFactory` can also be used as an alias for the singleton factory returned by the operation `DDS_DomainParticipantFactory_get_instance`.

The interface description of this class is as follows:

```
/*
 * interface DDS_DomainParticipantFactory
 */
/*
 * implemented API operations
 */
DDS_DomainParticipantFactory
DDS_DomainParticipantFactory_get_instance
```

```

        (void);
    DDS_DomainParticipant
        DDS_DomainParticipantFactory_create_participant
            (DDS_DomainParticipantFactory _this,
             const DomainId_t domainId,
             const DDS_DomainParticipantQos *qos,
             const struct DDS_DomainParticipantListener *a_listener,
             const DDS_StatusMask mask);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_delete_participant
            (DDS_DomainParticipantFactory _this,
             const DDS_DomainParticipant a_participant);
    DDS_DomainParticipant
        DDS_DomainParticipantFactory_lookup_participant
            (DDS_DomainParticipantFactory _this,
             const DomainId_t domainId);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_set_default_participant_qos
            (DDS_DomainParticipantFactory _this,
             const DDS_DomainParticipantQos *qos);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_get_default_participant_qos
            (DDS_DomainParticipantFactory _this,
             DDS_DomainParticipantQos *qos);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_set_qos
            (DDS_DomainParticipantFactory _this,
             const DDS_DomainParticipantFactoryQos *qos);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_get_qos
            (DDS_DomainParticipantFactory _this,
             DDS_DomainParticipantFactoryQos *qos);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_delete_domain
            (DDS_DomainParticipantFactory _this,
             DDS_Domain a_domain);
    DDS_Domain
        DDS_DomainParticipantFactory_lookup_domain
            (DDS_DomainParticipantFactory _this,
             const DomainId_t domainId);
    DDS_ReturnCode_t
        DDS_DomainParticipantFactory_delete_contained_entities
            (DDS_DomainParticipantFactory _this);

```

The following paragraphs describe the usage of all DDS_DomainParticipantFactory operations.

3.2.2.1 DDS_DomainParticipantFactory_create_participant

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_DomainParticipantFactory_create_participant
        (DDS_DomainParticipantFactory _this,
         const DomainId_t domainId,
         const DDS_DomainParticipantQos *qos,
         const struct DDS_DomainParticipantListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation creates a new DDS_DomainParticipant which will join the domain identified by domainId, with the desired DDS_DomainParticipantQos and attaches the optionally specified DDS_DomainParticipantListener to it.

Parameters

- in DDS_DomainParticipantFactory _this* - the DDS_DomainParticipantFactory object on which the operation is operated.
- in const DomainId_t domainId* - the ID of the Domain to which the DDS_DomainParticipant is joined. This should be a URI to the location of the configuration file that identifies the configuration details of the Domain, or the Domain name as specified in the configuration file. The actual ID used will also be applicable for the lookup_participant and get_domain_id operations; it is not allowed to mix URI and Domain name in operations on one DomainParticipant.
- in const DDS_DomainParticipantQos *qos* - a DDS_DomainParticipantQos for the new DDS_DomainParticipant. When this set of QosPolicy settings is inconsistent, no DDS_DomainParticipant is created.
- in const struct DDS_DomainParticipantListener *a_listener* - a pointer to the DDS_DomainParticipantListener instance which will be attached to the new DDS_DomainParticipant. It is permitted to use DDS_OBJECT_NIL as the value of the listener: this behaves as a DDS_DomainParticipantListener whose operations perform no action.
- in const DDS_StatusMask mask* - a bit-mask in which each bit enables the invocation of the DDS_DomainParticipantListener for a certain status.

Return Value

DDS_DomainParticipant -Return value is a pointer to the newly created *DDS_DomainParticipant*. In case of an error, the *DDS_OBJECT_NIL* pointer is returned.

Detailed Description

This operation creates a new *DDS_DomainParticipant*, with the desired *DDS_DomainParticipantQos* and attaches the optionally specified *DDS_DomainParticipantListener* to it. The *DDS_DomainParticipant* signifies that the calling application intends to join the Domain identified by the *domainId* argument.

If the specified *QosPolicy* settings are not consistent, the operation will fail; no *DDS_DomainParticipant* is created and the operation returns the *DDS_OBJECT_NIL* pointer. To delete the *DDS_DomainParticipant* the operation *DDS_DomainParticipantFactory_delete_participant* must be used.

Identifying the Domain

The *DDS_DomainParticipant* will attach to the Domain that is specified by the *domainId* parameter. This parameter consists of a string that represents either a URI to the location of the configuration file (e.g. "file:///projects/DDS/ospl.xml"), or the Domain name as specified in the configuration file. Note that to make multiple connections to a Domain (create multiple Participants for the same Domain) within a single process, all of the Participants must use the same identification (*i.e.* all use the URI or all use the Domain name). The configuration file identified by the URI specifies all configuration details of the Domain to which it refers. See the *Deployment Guide* for further details about the contents of this configuration file.

A NULL pointer may be assigned to the *DomainId*: in that case the location of the configuration file is extracted from the environment variable called *OSPL_URI*. This variable will be initialized when you source the *release.com* script (on platforms to which that applies) or, on the Windows platform, when you install the OpenSplice product. Initially it will point to the default configuration file that comes with OpenSplice, but of course you are free to change this to any configuration file that you want.

It is recommended to use this *OSPL_URI* variable instead of hard-coding the URI into your application, since this gives you much more flexibility in the deployment phase of your product.

Default QoS

The constant `DDS_PARTICIPANT_QOS_DEFAULT` can be used as parameter `qos` to create a `DDS_DomainParticipant` with the default `DDS_DomainParticipantQos` as set in the `DDS_DomainParticipantFactory`. The effect of using `DDS_PARTICIPANT_QOS_DEFAULT` is the same as calling the operation `DDS_DomainParticipantFactory_get_default_participant_qos` and using the resulting `DDS_DomainParticipantQos` to create the `DDS_DomainParticipant`.

Communication Status

For each communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever that communication status changes. For each communication status activated in the mask, the associated `DDS_DomainParticipantListener` operation is invoked and the communication status is reset to `FALSE`, as the listener implicitly accesses the status which is passed as a parameter to that operation. The fact that the status is reset prior to calling the listener means that if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset.

The following statuses are applicable to the `DDS_DomainParticipantListener`:

- `DDS_INCONSISTENT_TOPIC_STATUS` *(propagated)*
- `DDS_OFFERED_DEADLINE_MISSED_STATUS` *(propagated)*
- `DDS_REQUESTED_DEADLINE_MISSED_STATUS` *(propagated)*
- `DDS_OFFERED_INCOMPATIBLE_QOS_STATUS` *(propagated)*
- `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` *(propagated)*
- `DDS_SAMPLE_LOST_STATUS` *(propagated)*
- `DDS_SAMPLE_REJECTED_STATUS` *(propagated)*
- `DDS_DATA_ON_READERS_STATUS` *(propagated)*
- `DDS_DATA_AVAILABLE_STATUS` *(propagated)*
- `DDS_LIVELINESS_LOST_STATUS` *(propagated)*
- `DDS_LIVELINESS_CHANGED_STATUS` *(propagated)*
- `DDS_PUBLICATION_MATCHED_STATUS` *(propagated)*
- `DDS_SUBSCRIPTION_MATCHED_STATUS` *(propagated)*.



Be aware that the `DDS_PUBLICATION_MATCHED_STATUS` and `DDS_SUBSCRIPTION_MATCHED_STATUS` are not applicable when the infrastructure does not have the information available to determine connectivity.

This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_OBJECT_NIL`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

The Data Distribution Service will trigger the most specific and relevant `Listener`. In other words, in case a communication status is also activated on the `Listener` of a contained entity, the `Listener` on that contained entity is invoked instead of the `DDS_DomainParticipantListener`. This means that a status change on a contained entity only invokes the `DDS_DomainParticipantListener` if the contained entity itself does not handle the trigger event generated by the status change.

The statuses `DDS_DATA_ON_READERS_STATUS` and `DDS_DATA_AVAILABLE_STATUS` are “Read Communication Statuses” and are an exception to all other plain communication statuses: they have no corresponding status structure that can be obtained with a `get_<status_name>_status` operation and they are mutually exclusive. When new information becomes available to a `DataReader`, the Data Distribution Service will first look in an attached and activated `DDS_SubscriberListener` or `DDS_DomainParticipantListener` (in that order) for the `DDS_DATA_ON_READERS_STATUS`. In case the `DDS_DATA_ON_READERS_STATUS` can not be handled, the Data Distribution Service will look in an attached and activated `DDS_DataReaderListener`, `DDS_SubscriberListener` or `DDS_DomainParticipantListener` for the `DDS_DATA_AVAILABLE_STATUS` (in that order).

3.2.2.2 `DDS_DomainParticipantFactory_delete_participant`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipantFactory_delete_participant
        (DDS_DomainParticipantFactory _this,
         const DDS_DomainParticipant a_participant);
```


Description

This operation deletes a `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipantFactory _this` - the `DDS_DomainParticipantFactory` object on which the operation is operated.

in const `DDS_DomainParticipant a_participant` - a pointer to the `DDS_DomainParticipant`, which is to be deleted.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_OUT_OF_RESOURCES` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation deletes a `DDS_DomainParticipant`. A `DDS_DomainParticipant` cannot be deleted when it has any attached `DDS_Entity` objects. When the operation is called on a `DDS_DomainParticipant` with existing `DDS_Entity` objects, the operation returns `DDS_RETCODE_PRECONDITION_NOT_MET`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_DomainParticipant` is deleted.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `a_participant` is not a valid `DDS_DomainParticipant`.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the `DDS_DomainParticipant` contains one or more `DDS_Entity` objects.

3.2.2.3 `DDS_DomainParticipantFactory_get_default_participant_qos`

Synopsis

```
#include <dds_dcps.h>
```

```
DDS_ReturnCode_t
DDS_DomainParticipantFactory_get_default_participant_qos
(DDS_DomainParticipantFactory _this,
 DDS_DomainParticipantQos *qos);
```

Description

This operation gets the default DDS_DomainParticipantQos of the DDS_DomainParticipant.

Parameters

in `DDS_DomainParticipantFactory _this` - the DDS_DomainParticipantFactory object on which the operation is operated.

inout `DDS_DomainParticipantQos *qos` - a pointer to the DDS_DomainParticipantQos struct (provided by the application) in which the default DDS_DomainParticipantQos for the DDS_DomainParticipant is written.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation gets the default DDS_DomainParticipantQos of the DDS_DomainParticipant (that is the struct with the QosPolicy settings) which is used for newly created DDS_DomainParticipant objects, in case the constant DDS_PARTICIPANT_QOS_DEFAULT is used. The default DDS_DomainParticipantQos is only used when the constant is supplied as parameter qos to specify the DDS_DomainParticipantQos in the DDS_DomainParticipantFactory_create_participant operation. The application must provide the DDS_DomainParticipantQos struct in which the QosPolicy settings can be stored and provide a pointer to the struct. The operation writes the default QosPolicy settings to the struct pointed to by qos. Any settings in the struct are overwritten.

The values retrieved by this operation match the set of values specified on the last successful call to DDS_DomainParticipantFactory_set_default_participant_qos, or, if the call was never made, the default values as specified for each QosPolicy setting as defined in Table 5 on page 59.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the default *DDS_DomainParticipantQosPolicy* settings of this *DDS_DomainParticipantFactory* have successfully been copied into the specified *DDS_DomainParticipantQos* parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.2.4 *DDS_DomainParticipantFactory_get_instance*

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipantFactory
    DDS_DomainParticipantFactory_get_instance
        (void);
```

Description

This operation returns the *DDS_DomainParticipantFactory* singleton.

Parameters

<none>

Return Value

DDS_DomainParticipantFactory - return value is a pointer to the *DDS_DomainParticipantFactory*.

Detailed Description

This operation returns the *DDS_DomainParticipantFactory* singleton. The operation can be called multiple times without side-effects and it returns the same *DDS_DomainParticipantFactory* instance.

The pre-defined value *DDS_TheParticipantFactory* can also be used as an alias for the singleton factory returned by the operation *DDS_DomainParticipantFactory_get_instance*.

3.2.2.5 *DDS_DomainParticipantFactory_get_qos*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipantFactory_get_qos
        (DDS_DomainParticipantFactory _this,
```

```
DDS_DomainParticipantFactoryQos *qos);
```

Description

This operation allows access to the existing set of QoS policies for a DDS_DomainParticipantFactory.

Parameters

in `DDS_DomainParticipantFactory _this` - the DDS_DomainParticipantFactory object on which the operation is operated.

inout `DDS_DomainParticipantFactoryQos *qos` - a pointer to the destination DDS_DomainParticipantFactoryQos struct in which the QoSPolicy settings will be copied.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation allows access to the existing set of QoS policies of a DDS_DomainParticipantFactory on which this operation is used. This DDS_DomainParticipantFactoryQos is stored at the location pointed to by the qos parameter.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of QoS policy values applied to this DDS_DomainParticipantFactory has successfully been copied into the specified DDS_DomainParticipantFactoryQos parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.2.2.6 DDS_DomainParticipantFactory_lookup_participant

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
```

```
DDS_DomainParticipantFactory_lookup_participant
(DDS_DomainParticipantFactory _this,
 const DomainId_t domainId);
```

Description

This operation retrieves a previously created `DDS_DomainParticipant` belonging to the specified `domainId`.

Parameters

in `DDS_DomainParticipantFactory _this` - the `DDS_DomainParticipantFactory` object on which the operation is operated.

in `const DomainId_t domainId` - the ID of the Domain for which a joining `DomainParticipant` should be retrieved. This should be either a URI to the location of the configuration file that identifies the configuration details of the Domain., or the Domain name as specified in the configuration file. The actual value to be used (URI or Domain name) is dependent of the value used when creating the `DomainParticipant`, also see the `DDS_DomainParticipantFactory_create_participant` operation. If a `DomainParticipant` is created using the Domain name then the Domain name should be used to lookup the `DomainParticipant`, and *vice versa*: when created using a URI then the URI must be used with this operation.

Return Value

`DDS_DomainParticipant` - Return value is a pointer to the `DDS_DomainParticipant` retrieved. When no such `DDS_DomainParticipant` is found, the `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation retrieves a previously created `DDS_DomainParticipant` belonging to the specified `domainId`. If no such `DDS_DomainParticipant` exists, the operation will return `DDS_OBJECT_NIL`.

The `domainId` used to search for a specific `DDS_DomainParticipant` must be identical to the `domainId` that was used to create that specific `DDS_DomainParticipant`: a `NULL` pointer will not be resolved on this level. This means that a `DDS_DomainParticipant` that was created using a `domainId` set to `NULL` will not be found if you try to look it up using a hard-coded URI that has the same contents as the environment variable `OSPL_URI`.

If multiple `DDS_DomainParticipant` entities belonging to the specified `domainId` exist, then the operation will return one of them. It is not specified which one.

3.2.2.7 `DDS_DomainParticipantFactory_set_default_participant_qos`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipantFactory_set_default_participant_qos
        (DDS_DomainParticipantFactory _this,
         const DDS_DomainParticipantQos *qos);
```

Description

This operation sets the default `DDS_DomainParticipantQos` of the `DDS_DomainParticipant`.

Parameters

in `DDS_DomainParticipantFactory _this` - the `DDS_DomainParticipantFactory` object on which the operation is operated.

in *const* `DDS_DomainParticipantQos *qos` - the `DDS_DomainParticipantQos` struct, which contains the new default `DDS_DomainParticipantQos` for the newly created `DDS_DomainParticipants`.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: `DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation sets the default `DDS_DomainParticipantQos` of the `DDS_DomainParticipant` (that is the struct with the `QosPolicy` settings) which is used for newly created `DDS_DomainParticipant` objects, in case the constant `DDS_PARTICIPANT_QOS_DEFAULT` is used. The default `DDS_DomainParticipantQos` is only used when the constant is supplied as parameter `qos` to specify the `DDS_DomainParticipantQos` in the `DDS_DomainParticipantFactory_create_participant` operation. The

DDS_DomainParticipantQos is always self consistent, because its policies do not depend on each other. This means this operation never returns the DDS_RETCODE_INCONSISTENT_POLICY.

The values set by this operation are returned by DDS_DomainParticipantFactory_get_default_participant_qos.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new default DDS_DomainParticipantQos is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter qos is not a valid DDS_DomainParticipantQos. It contains a QosPolicy setting with a sequence that has inconsistent memory settings.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.2.8 DDS_DomainParticipantFactory_set_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipantFactory_set_qos
        (DDS_DomainParticipantFactory _this,
         const DDS_DomainParticipantFactoryQos *qos);
```

Description

This operation replaces the existing set of QosPolicy settings for a DDS_DomainParticipantFactory.

Parameters

- in* *DDS_DomainParticipantFactory _this* - is the DDS_DomainParticipantFactory object on which the operation is operated.
- in* *const DDS_DomainParticipantFactoryQos *qos* - must contain the new set of QosPolicy settings for the DDS_DomainParticipantFactory.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation replaces the existing set of *QosPolicy* settings for a *DDS_DomainParticipantFactory*. The parameter *qos* must contain the struct with the *QosPolicy* settings.

The set of *QosPolicy* settings specified by the *qos* parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned *DDS_RETCODE_OK*).

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new *DDS_DomainParticipantFactoryQos* is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.2.9 *DDS_DomainParticipantFactory_delete_domain*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DomainParticipantFactory_delete_domain
(DDS_DomainParticipantFactory _this,
 DDS_Domain a_domain);
```

Description

This operation deletes a *DDS_Domain*.

Parameters

in DDS_DomainParticipantFactory _this -
 the *DDS_DomainParticipantFactory* object on which the operation is operated.

in DDS_Domain a_domain - a pointer to the *DDS_Domain*, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation deletes a DDS_Domain.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_Domain is deleted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *a_domain* is not a valid DDS_Domain.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.2.10 DDS_DomainParticipantFactory_lookup_domain

Synopsis

```
#include <dds_dcps.h>
DDS_Domain
    DDS_DomainParticipantFactory_lookup_domain
        (DDS_DomainParticipantFactory _this,
         const DomainId_t domainId);
```

Description

This operation retrieves a previously created DDS_Domain proxy belonging to the specified domainId or creates a new DDS_Domain if no DDS_Domain yet exists but the Domain itself is available.

Parameters

in DDS_DomainParticipantFactory _this -
 the DDS_DomainParticipantFactory object on which the operation is operated.

in const DomainId_t domainId - the ID of the Domain for which a DDS_Domain proxy should be retrieved. This should be a URI to the location of the configuration file that identifies the configuration details of the Domain.

Return Value

DDS_Domain - Return value is a pointer to the DDS_Domain proxy retrieved. When no such DDS_Domain proxy is found or could be created, the DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation retrieves a previously created DDS_Domain proxy belonging to the specified domainId or creates a new DDS_Domain proxy if no DDS_Domain proxy was found, but the DomainId does refer to a valid Domain. If no such DDS_Domain exists or could be created, the operation will return DDS_OBJECT_NIL.

The domainId used to search for a specific DDS_Domain must not be a NULL pointer, as a NULL pointer will not be resolved on this level and in effect has no meaning.

3.2.2.11 DDS_DomainParticipantFactory_delete_contained_entities

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DomainParticipantFactory_delete_contained_entities
        (DDS_DomainParticipantFactory _this);
```

Description

This operation deletes all of the DDS_Entity objects that were created on the DDS_DomainParticipantFactory.

Parameters

in DDS_DomainParticipantFactory _this - the DDS_DomainParticipantFactory object on which the operation is performed.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:

DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_BAD_PARAMETER.

Detailed Description

This operation deletes all of the `DDS_Entity` objects that were created on the `DDS_DomainParticipantFactory` (it deletes all contained `DDS_DomainParticipant` objects). Prior to deleting each contained `DDS_Entity`, this operation regressively calls the `DDS_DomainParticipant_delete_contained_entities` operation on each `DDS_Participant`. In other words, this operation cleans up all `DDS_Entity` objects in the process.



NOTE: The operation will return `DDS_PRECONDITION_NOT_MET` if the any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained `DDS_DataReader` cannot be deleted because the application has called a `read` or `take` operation and has not called the corresponding `return_loan` operation to return the loaned samples. In such cases, the operation does not roll back any entity deletions performed prior to the detection of the problem.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - all contained `DDS_Entity` objects are deleted.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one or more of the contained entities are in a state where they cannot be deleted.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `_this` is not a valid `DDS_DomainParticipantFactory`.

3.2.3 Class `DDS_Domain`

The purpose of this class is to represent the `Domain` and allow certain `Domain`-wide operations to be performed. In essence it is a proxy to the `Domain`.

A `Domain` is a distributed concept that links all the applications that must be able to communicate with each other. It represents a communication plane: only the `DDS_Publishers` and the `DDS_Subscribers` attached to the same `Domain` can interact.

This class currently implements one function:

- It allows for a snapshot to be taken of all persistent data available within this Domain on local node level.

The interface description of this class is as follows:

```
/*
 * interface DDS_Domain
 */
DDS_ReturnCode_t
DDS_Domain_create_persistent_snapshot(
    DDS_Domain _this,
    const DDS_char* partition_expression,
    const DDS_char* topic_expression,
    const DDS_char* URI);
```

The following sections describe the usage of all DDS_Domain operations.

3.2.3.1 DDS_Domain_create_persistent_snapshot

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_Domain_create_persistent_snapshot(
    DDS_Domain _this,
    const DDS_char* partition_expression,
    const DDS_char* topic_expression,
    const DDS_char* URI);
```

Description

This operation will create a snapshot of all persistent data matching the provided partition and topic expressions and store the snapshot at the location indicated by the URI. Only persistent data available on the local node is considered.

Parameters

in DDS_Domain _this - the DDS_Domain object on which the operation is operated.

in DDS_char partition_expression* - the expression of all partitions involved in the snapshot; this may contain wildcards.

in DDS_char topic_expression* - the expression of all topics involved in the snapshot; this may contain wildcards.

in DDS_char uri* - the location where to store the snapshot. Currently only directories are supported.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation will create a snapshot of all persistent data matching the provided partition and topic expressions and store the snapshot at the location indicated by the URI. Only persistent data available on the local node is considered. This operation will fire an event to trigger the snapshot creation by the durability service and then return while the durability service fulfills the snapshot request; if no durability service is available then there is no persistent data available and the operation will return OK as a snapshot of an empty store is an empty store.

The created snapshot can then be used as the persistent store for the durability service next time it starts up by configuring the location of the snapshot as the persistent store in the configuration file. The durability service will then use the snapshot as the regular store (and can thus also alter its contents).

Return Code

When the operation returns:

- *DDS_RETCODE_OK* – The persistent snapshot is (being) created.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *partition_expression*, *topic_expression* or *uri* is NIL.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Domain proxy has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.2.4 DDS_DomainParticipantListener interface

Since a *DDS_DomainParticipant* is a *DDS_Entity*, it has the ability to have a *Listener* associated with it. In this case, the associated *Listener* should be of type *DDS_DomainParticipantListener*. This interface must be implemented by the application. A user-defined class must be provided by the application which must extend from the *DDS_DomainParticipantListener* class. **All**

DDS_DomainParticipantListener operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.



All operations for this interface must be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.

The DDS_DomainParticipantListener provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a QoSPolicy setting, etc. The DDS_DomainParticipantListener is related to changes in communication status DDS_StatusConditions.

The interface description of this class is as follows:

```
/*
 * interface DDS_DomainParticipantListener
 */
/*
 * inherited from DDS_TopicListener
 */
/* void
 *     DDS_DomainParticipantListener_on_inconsistent_topic
 *     (void *listener_data,
 *      DDS_Topic the_topic,
 *      const DDS_InconsistentTopicStatus *status);
 */
/*
 * inherited from DDS_PublisherListener
 */
/* void
 * DDS_DomainParticipantListener_on_offered_deadline_missed
 *     (void *listener_data,
 *      DDS_DataWriter writer,
 *      const DDS_OfferedDeadlineMissedStatus *status);
 */
/* void
 *     DDS_DomainParticipantListener_on_offered_incompatible_qos
 *     (void *listener_data,
 *      DDS_DataWriter writer,
 *      const DDS_OfferedIncompatibleQosStatus *status);
 */
/* void
 *     DDS_DomainParticipantListener_on_liveliness_lost
 *     (void *listener_data,
 *      DDS_DataWriter writer,
```

```

        *          const DDS_LivelinessLostStatus *status);
    */

/* void
 *   DDS_DomainParticipantListener_on_publication_matched
 *   (void *listener_data,
 *     DDS_DataWriter writer,
 *     const DDS_PublicationMatchedException *status);
 */
/*
 * inherited from DDS_SubscriberListener
 */
/* void
 *   DDS_DomainParticipantListener_on_data_on_readers
 *   (void *listener_data,
 *     DDS_Subscriber subs);
 */
/* void
 *   DDS_DomainParticipantListener_on_requested_deadline_missed
 *   (void *listener_data,
 *     DDS_DataReader reader,
 *     const DDS_RequestedDeadlineMissedException *status);
 */

/* void
 *   DDS_DomainParticipantListener_on_requested_incompatible_qos
 *   (void *listener_data,
 *     DDS_DataReader reader,
 *     const DDS_RequestedIncompatibleQosStatus *status);
 */

/* void
 *   DDS_DomainParticipantListener_on_sample_rejected
 *   (void *listener_data,
 *     DDS_DataReader reader,
 *     const DDS_SampleRejectedStatus *status);
 */

/* void
 *   DDS_DomainParticipantListener_on_liveliness_changed
 *   (void *listener_data,
 *     DDS_DataReader reader,
 *     const DDS_LivelinessChangedStatus *status);
 */

/* void
 *   DDS_DomainParticipantListener_on_data_available
 *   (void *listener_data,
 *     DDS_DataReader reader);
 */

```

```

/* void
 *   DDS_DomainParticipantListener_on_subscription_matched
 *   (void *listener_data,
 *       DDS_DataReader reader,
 *       const DDS_SubscriptionMatchedStatus *status);
 */

/* void
 *   DDS_DomainParticipantListener_on_sample_lost
 *   (void *listener_data,
 *       DDS_DataReader reader,
 *       const DDS_SampleLostStatus *status);
 */
/*
 * implemented API operations
 */
DDS_DomainParticipantListener
DDS_DomainParticipantListener__alloc
(void);

```

The next paragraphs list all `DDS_DomainParticipantListener` operations. Since these operations are all inherited, they are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.2.4.1 `DDS_DomainParticipantListener__alloc`

Synopsis

```

#include <dds_dcps.h>
DDS_DomainParticipantListener
DDS_DomainParticipantListener__alloc
(void);

```

Description

This operation creates a new `DDS_DomainParticipantListener`.

Parameters

<none>

Return Value

DDS_DomainParticipantListener - Return value is the handle to the newly created `DDS_DomainParticipantListener`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_DomainParticipantListener`. The `DDS_DomainParticipantListener` must be created using this operation. In other words, the application is not allowed to declare an object of type `DDS_DomainParticipantListener`. When the application wants to release the `DDS_DomainParticipantListener` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the `DDS_DomainParticipantListener`, a `DDS_OBJECT_NIL` pointer is returned instead.

3.2.4.2 `DDS_DomainParticipantListener_on_data_available` (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_data_available
        (void *listener_data,
         DDS_DataReader reader);
```

3.2.4.3 `DDS_DomainParticipantListener_on_data_on_readers` (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_SubscriberListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_data_on_readers
        (void *listener_data,
         DDS_Subscriber subs);
```

3.2.4.4 `DDS_DomainParticipantListener_on_inconsistent_topic` (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_TopicListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_inconsistent_topic
```

```
(void *listener_data,
    DDS_Topic the_topic,
    const DDS_InconsistentTopicStatus *status);
```

3.2.4.5 **DDS_DomainParticipantListener_on_liveliness_changed (inherited, abstract)**

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_liveliness_changed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_LivelinessChangedStatus *status);
```

3.2.4.6 **DDS_DomainParticipantListener_on_liveliness_lost (inherited, abstract)**

This operation is inherited and therefore not described here. See the class `DDS_DataWriterListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_liveliness_lost
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_LivelinessLostStatus *status);
```

3.2.4.7 **DDS_DomainParticipantListener_on_offered_deadline_missed (inherited, abstract)**

This operation is inherited and therefore not described here. See the class `DDS_DataWriterListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_offered_deadline_missed
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_OfferedDeadlineMissedStatus *status);
```

3.2.4.8 **DDS_DomainParticipantListener_on_offered_incompatible_qos**

(inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataWriterListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_offered_incompatible_qos
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_OfferedIncompatibleQosStatus *status);
```

3.2.4.9 DDS_DomainParticipantListener_on_publication_matched (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataWriterListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_publication_matched
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_PublicationMatchedException *status);
```

3.2.4.10 DDS_DomainParticipantListener_on_requested_deadline_missed (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_requested_deadline_missed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedDeadlineMissedException *status);
```

3.2.4.11 DDS_DomainParticipantListener_on_requested_incompatible_qos (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_requested_incompatible_qos
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedIncompatibleQosStatus *status);
```

3.2.4.12 DDS_DomainParticipantListener_on_sample_lost (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_sample_lost
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleLostStatus *status);
```

NOTE: This operation is not yet supported. It is scheduled for a future release.

3.2.4.13 DDS_DomainParticipantListener_on_sample_rejected (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_sample_rejected
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleRejectedStatus *status);
```

3.2.4.14 DDS_DomainParticipantListener_on_subscription_matched (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DomainParticipantListener_on_subscription_matched
        (void *listener_data,
         DDS_DataReader reader,
```

```
const DDS_SubscriptionMatchedStatus *status);
```

3.3 Topic-Definition Module

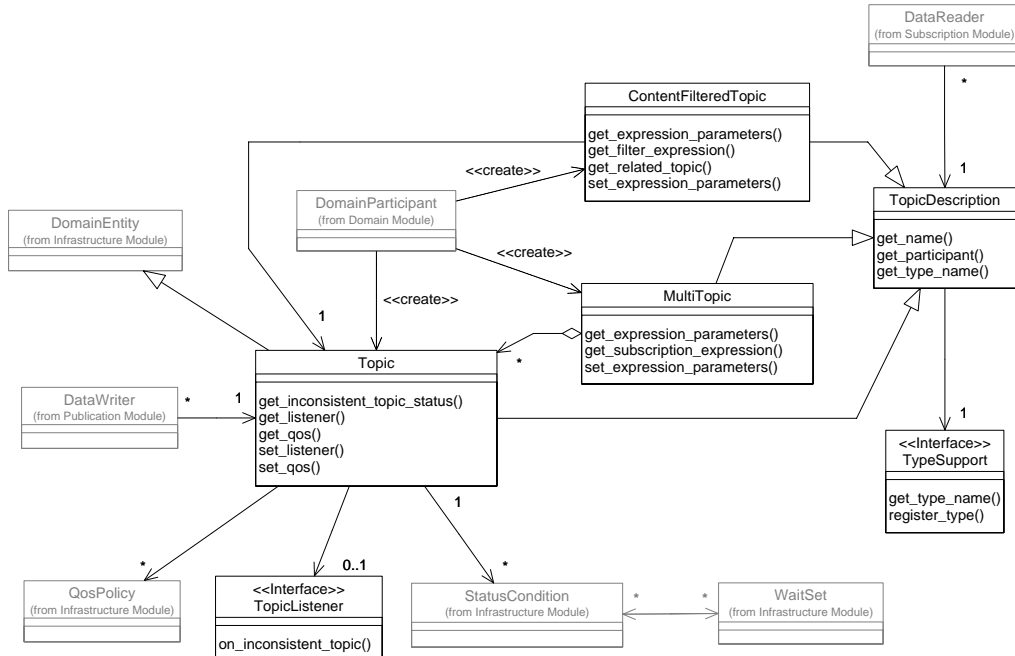


Figure 16 DCPS Topic-Definition Module Class Model

This module contains the following classes:

- DDS_TopicDescription (abstract)
- DDS_Topic
- DDS_ContentFilteredTopic
- DDS_MultiTopic
- DDS_TopicListener (interface)
- Topic-Definition type specific classes.

“Topic-Definition type specific classes” contains the generic class and the generated data type specific classes. For each data type, a data type specific class `<NameSpace>_<type>TypeSupport` is generated (based on IDL) by calling the pre-processor.

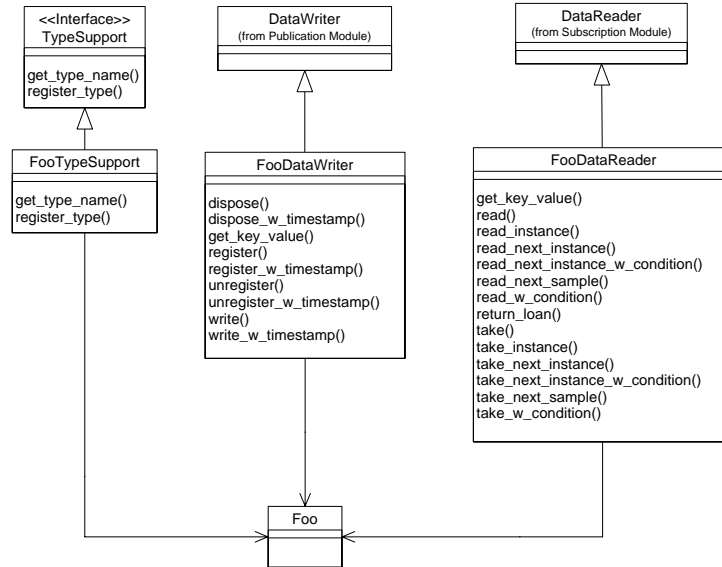


Figure 17 Pre-processor Generation of the Typed Classes for Data Type “Foo”

For instance, for the user-defined data type `Foo` (this also applies to other types), defined in the module `SPACE`; “Topic-Definition type specific classes” contains the following classes:

- `DDS_TypeSupport` (abstract)
- `SPACE_FooTypeSupport`.

`DDS_Topic` objects conceptually fit between publications and subscriptions. Publications must be known in such a way that subscriptions can refer to them unambiguously. A `DDS_Topic` is meant to fulfil that purpose: it associates a name (unique in the Domain), a data type, and `DDS_TopicQos`s related to the data itself.

3.3.1 Class `DDS_TopicDescription` (abstract)

This class is an abstract class. It is the base class for `DDS_Topic`, `DDS_ContentFilteredTopic` and `DDS_MultiTopic`.

The `DDS_TopicDescription` attribute `type_name` defines an unique data type that is made available to the Data Distribution Service via the `DDS_TypeSupport`. `DDS_TopicDescription` has also a name that allows it to be retrieved locally.

The interface description of this class is as follows:

```

/*
 * interface DDS_TopicDescription
 */
/*
 * implemented API operations

```

```

    */
    DDS_string
        DDS_TopicDescription_get_type_name
            (DDS_TopicDescription _this);

    DDS_string
        DDS_TopicDescription_get_name
            (DDS_TopicDescription _this);

    DDS_DomainParticipant
        DDS_TopicDescription_get_participant
            (DDS_TopicDescription _this);

```

The next paragraphs describe the usage of all DDS_TopicDescription operations.

3.3.1.1 DDS_TopicDescription_get_name

Synopsis

```

#include <dds_dcps.h>
DDS_string
    DDS_TopicDescription_get_name
        (DDS_TopicDescription _this);

```

Description

This operation returns the name used to create the DDS_TopicDescription.

Parameters

in DDS_TopicDescription _this - the DDS_TopicDescription object on which the operation is operated.

Return Value

DDS_string - the name of the DDS_TopicDescription.

Detailed Description

This operation returns the name used to create the DDS_TopicDescription.

3.3.1.2 DDS_TopicDescription_get_participant

Synopsis

```

#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_TopicDescription_get_participant
        (DDS_TopicDescription _this);

```

Description

This operation returns the `DDS_DomainParticipant` associated with the `DDS_TopicDescription` or the `DDS_OBJECT_NIL` pointer.

Parameters

in `DDS_TopicDescription _this` - the `DDS_TopicDescription` object on which the operation is operated.

Return Value

`DDS_DomainParticipant` - a pointer to the `DDS_DomainParticipant` associated with the `DDS_TopicDescription` or the `DDS_OBJECT_NIL` pointer.

Detailed Description

This operation returns the `DDS_DomainParticipant` associated with the `DDS_TopicDescription`. Note that there is exactly one `DDS_DomainParticipant` associated with each `DDS_TopicDescription`. When the `DDS_TopicDescription` was already deleted (there is no associated `DDS_DomainParticipant` any more), the `DDS_OBJECT_NIL` pointer is returned.

3.3.1.3 `DDS_TopicDescription_get_type_name`

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_TopicDescription_get_type_name
        (DDS_TopicDescription _this);
```

Description

This operation returns the registered name of the data type associated with the `DDS_TopicDescription`.

Parameters

in `DDS_TopicDescription _this` - the `DDS_TopicDescription` object on which the operation is operated.

Return Value

`DDS_string` - return value is the name of the data type of the `DDS_TopicDescription`.

Detailed Description

This operation returns the registered name of the data type associated with the `DDS_TopicDescription`.

3.3.2 Class `DDS_Topic`

`DDS_Topic` is the most basic description of the data to be published and subscribed.

A `DDS_Topic` is identified by its name, which must be unique in the whole Domain. In addition (by virtue of extending `DDS_TopicDescription`) it fully identifies the type of data that can be communicated when publishing or subscribing to the `DDS_Topic`.

`DDS_Topic` is the only `DDS_TopicDescription` that can be used for publications and therefore a specialized `DDS_DataWriter` is associated to the `DDS_Topic`.

The interface description of this class is as follows:

```
/*
 * interface DDS_Topic
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *   DDS_Topic_get_statuscondition
 *   (DDS_Topic _this);
 */
/* DDS_StatusMask
 *   DDS_Topic_get_status_changes
 *   (DDS_Topic _this);
 */
/* DDS_ReturnCode_t
 *   DDS_Topic_enable
 *   (DDS_Topic _this);
 */
/*
 * inherited from class DDS_TopicDescription
 */
/* DDS_string
 *   DDS_Topic_get_type_name
 *   (DDS_Topic _this);
 */

/* DDS_string
 *   DDS_Topic_get_name
 *   (DDS_Topic _this);
 */

/* DDS_DomainParticipant
```

```

*     DDS_Topic_get_participant
*     (DDS_Topic _this);
*/
/*
* implemented API operations
*/
DDS_ReturnCode_t
DDS_Topic_set_qos
    (DDS_Topic _this,
     const DDS_TopicQos *qos);
DDS_ReturnCode_t
DDS_Topic_get_qos
    (DDS_Topic _this,
     DDS_TopicQos *qos);
DDS_ReturnCode_t
DDS_Topic_set_listener
    (DDS_Topic _this,
     const struct DDS_TopicListener *a_listener,
     const DDS_StatusMask mask);
struct DDS_TopicListener
DDS_Topic_get_listener
    (DDS_Topic _this);
DDS_ReturnCode_t
DDS_Topic_get_inconsistent_topic_status
    (DDS_Topic _this,
     DDS_InconsistentTopicStatus *a_status);

```

The next paragraphs describe the usage of all DDS_Topic operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.3.2.1 DDS_Topic_enable (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Topic_enable
        (DDS_Topic _this);

```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.3.2.2 DDS_Topic_get_inconsistent_topic_status

Synopsis

```

#include <dds_dcps.h>

```

```

DDS_ReturnCode_t
    DDS_Topic_get_inconsistent_topic_status
        (DDS_Topic _this,
         DDS_InconsistentTopicStatus *a_status);

```

Description

This operation obtains the DDS_InconsistentTopicStatus of the DDS_Topic.

Parameters

in DDS_Topic _this - the DDS_Topic object on which the operation is operated.

inout DDS_InconsistentTopicStatus *a_status - the contents of the DDS_InconsistentTopicStatus struct of the DDS_Topic will be copied into the location specified by a_status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation obtains the DDS_InconsistentTopicStatus of the DDS_Topic. The DDS_InconsistentTopicStatus can also be monitored using a DDS_TopicListener or by using the associated DDS_StatusCondition.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the current DDS_InconsistentTopicStatus of this DDS_Topic has successfully been copied into the specified a_status parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Topic has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.3.2.3 DDS_Topic_get_listener

Synopsis

```
#include <dds_dcps.h>
```

```
struct DDS_TopicListener
    DDS_Topic_get_listener
        (DDS_Topic _this);
```

Description

This operation allows access to a DDS_TopicListener.

Parameters

in DDS_Topic _this - the DDS_Topic object on which the operation is operated.

Return Value

struct DDS_TopicListener - to the DDS_TopicListener attached to the DDS_Topic.

Detailed Description

This operation allows access to a DDS_TopicListener attached to the DDS_Topic. When no DDS_TopicListener was attached to the DDS_Topic, the DDS_OBJECT_NIL pointer is returned.

3.3.2.4 DDS_Topic_get_name (inherited)

This operation is inherited and therefore not described here. See the class DDS_TopicDescription for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_Topic_get_name
        (DDS_Topic _this);
```

3.3.2.5 DDS_Topic_get_participant (inherited)

This operation is inherited and therefore not described here. See the class DDS_TopicDescription for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_Topic_get_participant
        (DDS_Topic _this);
```

3.3.2.6 DDS_Topic_get_qos

Synopsis

```
#include <dds_dcps.h>
```

```

DDS_ReturnCode_t
    DDS_Topic_get_qos
        (DDS_Topic _this,
         DDS_TopicQos *qos);

```

Description

This operation allows access to the existing set of QoS policies for a `DDS_Topic`.

Parameters

in `DDS_Topic _this` - the `DDS_Topic` object on which the operation is operated.

inout `DDS_TopicQos *qos` - a pointer to the destination `DDS_TopicQos` struct in which the `QosPolicy` settings will be copied.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are: `DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation allows access to the existing set of QoS policies of a `DDS_Topic` on which this operation is used. This `DDS_TopicQos` is stored at the location pointed to by the `qos` parameter.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of QoS policy values applied to this `DDS_Topic` has successfully been copied into the specified `DDS_TopicQos` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Topic` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.3.2.7 `DDS_Topic_get_status_changes` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
    DDS_Topic_get_status_changes
        (DDS_Topic _this);
```

3.3.2.8 DDS_Topic_get_statuscondition (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
    DDS_Topic_get_statuscondition
        (DDS_Topic _this);
```

3.3.2.9 DDS_Topic_get_type_name (inherited)

This operation is inherited and therefore not described here. See the class `DDS_TopicDescription` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_Topic_get_type_name
        (DDS_Topic _this);
```

3.3.2.10 DDS_Topic_set_listener**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Topic_set_listener
        (DDS_Topic _this,
         const struct DDS_TopicListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation attaches a `DDS_TopicListener` to the `DDS_Topic`.

Parameters

in `DDS_Topic _this` - the `DDS_Topic` object on which the operation is operated.

in `const struct DDS_TopicListener *a_listener` - a pointer to the `DDS_TopicListener` instance, which will be attached to the `DDS_Topic`.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the `DDS_TopicListener` for a certain status.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation attaches a `DDS_TopicListener` to the `DDS_Topic`. Only one `DDS_TopicListener` can be attached to each `DDS_Topic`. If a `DDS_TopicListener` was already attached, the operation will replace it with the new one. When `a_listener` is the `DDS_OBJECT_NIL` pointer, it represents a listener that is treated as a NOOP¹ for all statuses activated in the bitmask.

Communication Status

For each communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever that plain communication status changes. For each plain communication status activated in the `mask`, the associated `DDS_TopicListener` operation is invoked and the communication status is reset to `FALSE`, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the `get_<status_name>` from inside the listener it will see the status already reset. An exception to this rule is the `DDS_OBJECT_NIL` listener, which does not reset the communication statuses for which it is invoked.

The following statuses are applicable to the `DDS_TopicListener`:

- `DDS_INCONSISTENT_TOPIC_STATUS`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

1. Short for **No-Operation**, an instruction that performs nothing at all.

Status Propagation

In case a communication status is not activated in the mask of the `DDS_TopicListener`, the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` and a `DDS_Topic` specific behaviour when needed. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_TopicListener` is attached.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Topic` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.3.2.11 `DDS_Topic_set_qos`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Topic_set_qos
        (DDS_Topic _this,
         const DDS_TopicQos *qos);
```

Description

This operation replaces the existing set of `QosPolicy` settings for a `DDS_Topic`.

Parameters

in `DDS_Topic _this` - the `DDS_Topic` object on which the operation is operated.

in `const DDS_TopicQos *qos` - new set of `QosPolicy` settings for the `DDS_Topic`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_UNSUPPORTED`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_IMMUTABLE_POLICY` or `DDS_RETCODE_INCONSISTENT_POLICY`.

Detailed Description

This replaces the existing set of `QosPolicy` settings for a `DDS_Topic`. The parameter `qos` must contain the struct with the `QosPolicy` settings which is checked for self-consistency and mutability. When the application tries to change a `QosPolicy` setting for an enabled `DDS_Topic`, which can only be set before the `DDS_Topic` is enabled, the operation will fail and a `DDS_RETCODE_IMMUTABLE_POLICY` is returned. In other words, the application must provide the currently set `QosPolicy` settings in case of the immutable `QosPolicy` settings. Only the mutable `QosPolicy` settings can be changed. When `qos` contains conflicting `QosPolicy` settings (not self-consistent), the operation will fail and a `DDS_RETCODE_INCONSISTENT_POLICY` is returned.

The set of `QosPolicy` settings specified by the `qos` parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned `DDS_RETCODE_OK`).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the new `DDS_TopicQos` is set.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `qos` is not a valid `DDS_TopicQos`. It contains a `QosPolicy` setting with an invalid `DDS_Duration_t` value, an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- `DDS_RETCODE_UNSUPPORTED` - one or more of the selected `QosPolicy` values are currently not supported by OpenSplice.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Topic` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

- `DDS_RETCODE_IMMUTABLE_POLICY` - the parameter `qos` contains an immutable `QosPolicy` setting with a different value than set during enabling of the `DDS_Topic`.
- `DDS_RETCODE_INCONSISTENT_POLICY` - the parameter `qos` contains conflicting `QosPolicy` settings, e.g. a history depth that is higher than the specified resource limits.

3.3.3 Class `DDS_ContentFilteredTopic`

`DDS_ContentFilteredTopic` is a specialization of `DDS_TopicDescription` that allows for content based subscriptions.

`DDS_ContentFilteredTopic` describes a more sophisticated subscription that indicates the `DDS_Subscriber` does not necessarily want to see all values of each instance published under the `DDS_Topic`. Rather, it only wants to see the values whose contents satisfy certain criteria. Therefore this class must be used to request content-based subscriptions.

The selection of the content is done using the SQL based filter with parameters to adapt the filter clause.

Appendix H, *DCPS Queries and Filters* describes the syntax of the SQL based filter and the parameters.

The interface description of this class is as follows:

```

/*
 * interface DDS_ContentFilteredTopic
 */
/*
 * inherited from class DDS_TopicDescription
 */
/* DDS_string
 *   DDS_ContentFilteredTopic_get_type_name
 *   (DDS_ContentFilteredTopic _this);
 */
/* DDS_string
 *   DDS_ContentFilteredTopic_get_name
 *   (DDS_ContentFilteredTopic _this);
 */

/* DDS_DomainParticipant
 *   DDS_ContentFilteredTopic_get_participant
 *   (DDS_ContentFilteredTopic _this);
 */
/*
 * implemented API operations
 */
DDS_string
    DDS_ContentFilteredTopic_get_filter_expression

```

```

        (DDS_ContentFilteredTopic _this);

DDS_ReturnCode_t
    DDS_ContentFilteredTopic_get_expression_parameters
        (DDS_ContentFilteredTopic _this,
         DDS_StringSeq *expression_parameters);

DDS_ReturnCode_t
    DDS_ContentFilteredTopic_set_expression_parameters
        (DDS_ContentFilteredTopic _this,
         const DDS_StringSeq *expression_parameters);

DDS_Topic
    DDS_ContentFilteredTopic_get_related_topic
        (DDS_ContentFilteredTopic _this);
/*

```

The next paragraphs describe the usage of all DDS_ContentFilteredTopic operations.

3.3.3.1 DDS_ContentFilteredTopic_get_expression_parameters

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_ContentFilteredTopic_get_expression_parameters
        (DDS_ContentFilteredTopic _this,
         DDS_StringSeq *expression_parameters);

```

Description

This operation obtains the expression parameters associated with the DDS_ContentFilteredTopic.

Parameters

in *DDS_ContentFilteredTopic _this* - the DDS_ContentFilteredTopic object on which the operation is operated.

inout *DDS_StringSeq *expression_parameters* - a handle to a sequence of strings that will be used to store the parameters used in the SQL expression.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation obtains the expression parameters associated with the `DDS_ContentFilteredTopic`. That is, the parameters specified on the last successful call to `DDS_ContentFilteredTopic_set_expression_parameters`, or if `DDS_ContentFilteredTopic_set_expression_parameters` was never called, the parameters specified when the `DDS_ContentFilteredTopic` was created.

The resulting handle contains a sequence of strings with the parameters used in the SQL expression (i.e., the %n tokens in the expression). The number of parameters in the result sequence will exactly match the number of %n tokens in the filter expression associated with the `DDS_ContentFilteredTopic`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of expression parameters applied to this `DDS_ContentFilteredTopic` has successfully been copied into the specified `expression_parameters` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_ContentFilteredTopic` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.3.3.2 `DDS_ContentFilteredTopic_get_filter_expression`

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_ContentFilteredTopic_get_filter_expression
        (DDS_ContentFilteredTopic _this);
```

Description

This operation returns the `filter_expression` associated with the `DDS_ContentFilteredTopic`.

Parameters

in `DDS_ContentFilteredTopic _this` - the `DDS_ContentFilteredTopic` object on which the operation is operated.

Return Value

DDS_string - result is a handle to a string which holds the SQL filter expression.

Detailed Description

This operation returns the `filter_expression` associated with the `DDS_ContentFilteredTopic`. That is, the expression specified when the `DDS_ContentFilteredTopic` was created.

The filter expression result is a string that specifies the criteria to select the data samples of interest. It is similar to the `WHERE` clause of an SQL expression.

3.3.3.3 DDS_ContentFilteredTopic_get_name (inherited)

This operation is inherited and therefore not described here. See the class `DDS_TopicDescription` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_ContentFilteredTopic_get_name
        (DDS_ContentFilteredTopic _this);
```

3.3.3.4 DDS_ContentFilteredTopic_get_participant (inherited)

This operation is inherited and therefore not described here. See the class `DDS_TopicDescription` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_ContentFilteredTopic_get_participant
        (DDS_ContentFilteredTopic _this);
```

3.3.3.5 DDS_ContentFilteredTopic_get_related_topic**Synopsis**

```
#include <dds_dcps.h>
DDS_Topic
    DDS_ContentFilteredTopic_get_related_topic
        (DDS_ContentFilteredTopic _this);
```

Description

This operation returns the `DDS_Topic` associated with the `DDS_ContentFilteredTopic`.

Parameters

in DDS_ContentFilteredTopic _this - the DDS_ContentFilteredTopic object on which the operation is operated.

Return Value

DDS_Topic - result is a handle to the base topic on which the filtering will be applied.

Detailed Description

This operation returns the DDS_Topic associated with the DDS_ContentFilteredTopic. That is, the DDS_Topic specified when the DDS_ContentFilteredTopic was created. This DDS_Topic is the base topic on which the filtering will be applied.

3.3.3.6 DDS_ContentFilteredTopic_get_type_name (inherited)

This operation is inherited and therefore not described here. See the class DDS_TopicDescription for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_ContentFilteredTopic_get_type_name
        (DDS_ContentFilteredTopic _this);
```

3.3.3.7 DDS_ContentFilteredTopic_set_expression_parameters**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_ContentFilteredTopic_set_expression_parameters
        (DDS_ContentFilteredTopic _this,
         const DDS_StringSeq *expression_parameters);
```

Description

This operation changes the expression parameters associated with the DDS_ContentFilteredTopic.

Parameters

in DDS_ContentFilteredTopic _this - the DDS_ContentFilteredTopic object on which the operation is operated.

*in const DDS_StringSeq *expression_parameters* - the handle to a sequence of strings with the parameters used in the SQL expression (i.e., the number of %n tokens in the expression). The number of values in *expression_parameters* must be equal or greater than the highest referenced %n token in the *subscription_expression*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation changes the expression parameters associated with the *DDS_ContentFilteredTopic*. The parameter *expression_parameters* is a handle to a sequence of strings with the parameters used in the SQL expression. The number of values in *expression_parameters* must be equal or greater than the highest referenced %n token in the *filter_expression* (e.g. if %1 and %8 are used as parameter in the *filter_expression*, the *expression_parameters* should at least contain $n+1 = 9$ values). This is the filter expression specified when the *DDS_ContentFilteredTopic* was created.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new expression parameters are set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the number of parameters in *expression_parameters* does not match the number of “%n” tokens in the expression for this *DDS_ContentFilteredTopic* or one of the parameters is an illegal parameter.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_ContentFilteredTopic* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.3.4 Class DDS_MultiTopic

DDS_MultiTopic is a specialization of DDS_TopicDescription that allows subscriptions to combine, filter and/or rearrange data coming from several DDS_Topics.

DDS_MultiTopic allows a more sophisticated subscription that can select and combine data received from multiple DDS_Topics into a single data type (specified by the inherited type_name). The data will then be filtered (selection) and possibly re-arranged (aggregation and/or projection) according to an SQL expression with parameters to adapt the filter clause.

The interface description of this class is as follows:

```

/*
 * interface DDS_MultiTopic
 */
/*
 * inherited from class DDS_TopicDescription
 */
/* DDS_string
 *     DDS_MultiTopic_get_type_name
 *     (DDS_MultiTopic _this);
 */
/* DDS_string
 *     DDS_MultiTopic_get_name
 *     (DDS_MultiTopic _this);
 */

/* DDS_DomainParticipant
 *     DDS_MultiTopic_get_participant
 *     (DDS_MultiTopic _this);
 */
/*
 * implemented API operations
 */
    DDS_string
        DDS_MultiTopic_get_subscription_expression
            (DDS_MultiTopic _this);

    DDS_ReturnCode_t
        DDS_MultiTopic_get_expression_parameters
            (DDS_MultiTopic _this,
             DDS_StringSeq *expression_parameters);

    DDS_ReturnCode_t
        DDS_MultiTopic_set_expression_parameters
            (DDS_MultiTopic _this,
             const DDS_StringSeq *expression_parameters);

```


The next paragraphs describe the usage of all `DDS_MultiTopic` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

NOTE: `DDS_MultiTopic` operations have not been yet been implemented. Multitopic functionality is scheduled for a future release.

3.3.4.1 `DDS_MultiTopic_get_expression_parameters`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_MultiTopic_get_expression_parameters
        (DDS_MultiTopic _this,
         DDS_StringSeq *expression_parameters);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

Description

This operation returns the expression parameters associated with the `DDS_MultiTopic`.

Parameters

in `DDS_MultiTopic _this` - the `DDS_MultiTopic` object on which the operation is operated.

inout `DDS_StringSeq *expression_parameters` - a handle to a sequence of strings that will be used to store the parameters used in the SQL expression.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation obtains the expression parameters associated with the `DDS_MultiTopic`. That is, the parameters specified on the last successful call to `DDS_MultiTopic_set_expression_parameters`, or if `DDS_MultiTopic_set_expression_parameters` was never called, the parameters specified when the `DDS_MultiTopic` was created.

The resulting handle contains a sequence of strings with the values of the parameters used in the SQL expression (i.e., the %n tokens in the expression). The number of parameters in the result sequence will exactly match the number of %n tokens in the filter expression associated with the DDS_MultiTopic.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of expression parameters applied to this DDS_MultiTopic has successfully been copied into the specified `expression_parameters` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the DDS_MultiTopic has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.3.4.2 DDS_MultiTopic_get_name (inherited)

This operation is inherited and therefore not described here. See the class DDS_TopicDescription for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_MultiTopic_get_name
        (DDS_MultiTopic _this);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.3.4.3 DDS_MultiTopic_get_participant (inherited)

This operation is inherited and therefore not described here. See the class DDS_TopicDescription for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_MultiTopic_get_participant
        (DDS_MultiTopic _this);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.3.4.4 DDS_MultiTopic_get_subscription_expression

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_MultiTopic_get_subscription_expression
        (DDS_MultiTopic _this);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

Description

This operation returns the subscription expression associated with the DDS_MultiTopic.

Parameters

in DDS_MultiTopic _this - is the DDS_MultiTopic object on which the operation is operated.

Return Value

DDS_string - a handle to a string which holds the SQL subscription expression.

Detailed Description

This operation returns the subscription expression associated with the DDS_MultiTopic. That is, the expression specified when the DDS_MultiTopic was created.

The subscription expression result is a string that specifies the criteria to select the data samples of interest. In other words, it identifies the selection and rearrangement of data from the associated DDS_Topics. It is an SQL expression where the SELECT clause provides the fields to be kept, the FROM part provides the names of the DDS_Topics that are searched for those fields, and the WHERE clause gives the content filter. The DDS_Topics combined may have different types but they are restricted in that the type of the fields used for the NATURAL JOIN operation must be the same.

3.3.4.5 DDS_MultiTopic_get_type_name (inherited)

This operation is inherited and therefore not described here. See the class DDS_TopicDescription for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_MultiTopic_get_type_name
        (DDS_MultiTopic _this);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.3.4.6 DDS_MultiTopic_set_expression_parameters

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_MultiTopic_set_expression_parameters
        (DDS_MultiTopic _this,
         const DDS_StringSeq *expression_parameters);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

Description

This operation changes the expression parameters associated with the DDS_MultiTopic.

Parameters

in *DDS_MultiTopic _this* - the DDS_MultiTopic object on which the operation is operated.

in *const DDS_StringSeq *expression_parameters* - the handle to a sequence of strings with the parameters used in the SQL expression.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation changes the expression parameters associated with the DDS_MultiTopic. The parameter *expression_parameters* is a handle to a sequence of strings with the parameters used in the SQL expression. The number of parameters in *expression_parameters* must exactly match the number of %n tokens in the subscription expression associated with the DDS_MultiTopic. This is the subscription expression specified when the DDS_MultiTopic was created.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new expression parameters are set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the number of parameters in *expression_parameters* does not match the number of “%n” tokens in the expression for this *DDS_MultiTopic* or one of the parameters is an illegal parameter.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_MultiTopic* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.3.5 DDS_TopicListener Interface

Since a *DDS_Topic* is a *DDS_Entity*, it has the ability to have a *Listener* associated with it. In this case, the associated *Listener* should be of type *DDS_TopicListener*. This interface must be implemented by the application. A user-defined class must be provided by the application which must extend from the *DDS_TopicListener* class. **All** *DDS_TopicListener* operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.



NOTE: All operations for this interface must be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.

The *DDS_TopicListener* provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as an inconsistent *DDS_Topic*. The *DDS_TopicListener* is related to changes in communication status.

The interface description of this class is as follows:

```
/*
 * interface DDS_TopicListener
 */
/*
 * abstract external operations
 */
void
    DDS_TopicListener_on_inconsistent_topic
        (void *listener_data,
         DDS_Topic the_topic,
         const DDS_InconsistentTopicStatus *status);
/*
 * implemented API operations
 */
```

```

DDS_TopicListener
    DDS_TopicListener__alloc
        (void);

```

The next paragraph describes the usage of the `DDS_TopicListener` operation. This abstract operation is fully described since it must be implemented by the application.

3.3.5.1 DDS_TopicListener__alloc

Synopsis

```

#include <dds_dcps.h>
DDS_TopicListener
    DDS_TopicListener__alloc
        (void);

```

Description

This operation creates a new `DDS_TopicListener`.

Parameters

<none>

Return Value

DDS_TopicListener - the handle to the newly created `DDS_TopicListener`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_TopicListener`. The `DDS_TopicListener` must be created using this operation. In other words, the application is not allowed to declare an object of type `DDS_TopicListener`. When the application wants to release the `DDS_TopicListener` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the `DDS_TopicListener`, a `DDS_OBJECT_NIL` pointer is returned instead.

3.3.5.2 DDS_TopicListener_on_inconsistent_topic (abstract)

Synopsis

```

#include <dds_dcps.h>
void
    DDS_TopicListener_on_inconsistent_topic
        (void *listener_data,
         DDS_Topic the_topic,
         const DDS_InconsistentTopicStatus *status);

```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the `DDS_InconsistentTopicStatus` changes.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the `Listener`.

in DDS_Topic the_topic - a pointer to the `DDS_Topic` on which the conflict occurred (this is an input to the application).

*in const DDS_InconsistentTopicStatus *status* - the `DDS_InconsistentTopicStatus` struct (this is an input to the application).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the `DDS_InconsistentTopicStatus` changes. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant `DDS_TopicListener` is installed and enabled for the `DDS_InconsistentTopicStatus`. The `DDS_InconsistentTopicStatus` will change when another `DDS_Topic` exists with the same `topic_name` but different characteristics.

The Data Distribution Service will call the `DDS_TopicListener` operation with a parameter `the_topic`, which will contain a pointer to the `DDS_Topic` on which the conflict occurred and a parameter `status`, which will contain the `DDS_InconsistentTopicStatus` struct.

3.3.6 Topic-Definition Type Specific Classes

This paragraph describes the generic `DDS_TypeSupport` class and the derived application type specific `<NameSpace>_<type>TypeSupport` classes which together implement the application `DDS_Topic` interface. For each application type, used as `DDS_Topic` data type, the pre-processor generates a `<NameSpace>_<type>DataReader` class from an IDL type description. The `SPACE_FooTypeSupport` class that would be generated by the pre-processor for a fictional type `Foo` (defined in the module `SPACE`) describes the `<NameSpace>_<type>TypeSupport` classes.

3.3.6.1 Class DDS_TypeSupport (abstract)

The DDS_Topic, DDS_MultiTopic or DDS_ContentFilteredTopic is bound to a data type described by the type name argument. Prior to creating a DDS_Topic, DDS_MultiTopic or DDS_ContentFilteredTopic, the data type must have been registered with the Data Distribution Service. This is done using the data type specific DDS_TypeSupport_register_type operation on a derived class of the DDS_TypeSupport interface. A derived class is generated for each data type used by the application, by calling the pre-processor.

The interface description of this class is as follows:

```
/*
 * interface DDS_TypeSupport
 */
/*
 * abstract operations
 */
/* DDS_TypeSupport
 *   DDS_TypeSupport__alloc
 *       (void);
 */
/* DDS_ReturnCode_t
 *   DDS_TypeSupport_register_type
 *       (DDS_TypeSupport _this,
 *        Domainparticipant domain,
 *        DDS_string type_name);
 *
 * DDS_string
 *   DDS_TypeSupport_get_type_name
 *       (DDS_TypeSupport _this);
 */
/*
 * implemented API operations
 *   <no operations>
 */
```

The next paragraph list the DDS_TypeSupport operation. This abstract operation is listed but not fully described since it is not implemented in this class. The full description of this operation is given in the SPACE_FooTypeSupport class (for the data type example Foo), which contains the data type specific implementation of this operation.

3.3.6.2 DDS_TypeSupport__alloc (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>TypeSupport` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooTypeSupport` class.

Synopsis

```
#include <dds_dcps.h>
DDS_TypeSupport
    DDS_TypeSupport__alloc
        (void);
```

3.3.6.3 DDS_TypeSupport_get_type_name (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>TypeSupport` class. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooTypeSupport` class.

Synopsis

```
#include <dds_dcps.h>
DDS_string
    DDS_TypeSupport_get_type_name
        (DDS_TypeSupport _this);
```

3.3.6.4 DDS_TypeSupport_register_type (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>TypeSupport` class. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooTypeSupport` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_TypeSupport_register_type
        (DDS_TypeSupport _this,
         Domainparticipant domain,
         DDS_string type_name);
```

3.3.6.5 Class SPACE_FooTypeSupport

The pre-processor generates from IDL type descriptions the application `<NameSpace>_<type>TypeSupport` classes. For each application data type that is used as `DDS_Topic` data type, a typed class `<NameSpace>_<type>TypeSupport` is derived from the `DDS_TypeSupport`

class. In this paragraph, the class `SPACE_FooTypeSupport` describes the operations of these derived `<NameSpace>_<type>TypeSupport` classes as an example for the fictional application type `Foo` (defined in the module `SPACE`).

For instance, for an application, the definitions are located in the `Space.idl` file. The pre-processor will generate a `Space.h` include file.

The `DDS_Topic`, `DDS_MultiTopic` or `DDS_ContentFilteredTopic` is bound to a data type described by the type `type_name` argument. Prior to creating a `DDS_Topic`, `DDS_MultiTopic` or `DDS_ContentFilteredTopic`, the data type must have been registered with the Data Distribution Service. This is done using the data type specific `SPACE_FooTypeSupport_register_type` operation on the `<NameSpace>_<type>TypeSupport` class for each data type. A derived class is generated for each data type used by the application, by calling the pre-processor.

The interface description of this class is as follows:

```
/*
 * interface SPACE_FooTypeSupport
 */
/*
 * implemented API operations
 */
SPACE_FooTypeSupport
SPACE_FooTypeSupport__alloc
(void);
DDS_ReturnCode_t
SPACE_FooTypeSupport_register_type
(SPACE_FooTypeSupport _this,
 DDS_DomainParticipant domain,
 DDS_string type_name);
DDS_string
SPACE_FooTypeSupport_get_type_name
(SPACE_FooTypeSupport _this);
```

The next paragraph describes the usage of the `SPACE_FooTypeSupport` operations.

3.3.6.6 SPACE_FooTypeSupport__alloc

Synopsis

```
#include <Space.h>
SPACE_FooTypeSupport
SPACE_FooTypeSupport__alloc
(void);
```

Description

This operation creates a new `SPACE_FooTypeSupport`.

Parameters

<none>

Return Value

SPACE_FooTypeSupport - the handle to the newly created *SPACE_FooTypeSupport*. In case of an error, a nil pointer is returned.

Detailed Description

This operation creates a new *SPACE_FooTypeSupport*. The *SPACE_FooTypeSupport* must be created using this operation. In other words, the application is not allowed to declare an object of type *SPACE_FooTypeSupport*. When the application wants to release the *SPACE_FooTypeSupport* it must be released using *DDS_free*.

In case there are insufficient resources available to allocate the *SPACE_FooTypeSupport*, a nil pointer is returned instead.

3.3.6.7 *SPACE_FooTypeSupport_get_type_name*

Synopsis

```
#include <Space.h>
DDS_string
SPACE_FooTypeSupport_get_type_name
    (SPACE_FooTypeSupport _this);
```

Description

This operation returns the default name of the data type associated with the *SPACE_FooTypeSupport*.

Parameters

in *SPACE_FooTypeSupport _this* - the *SPACE_FooTypeSupport* object on which the operation is operated.

Return Value

DDS_string - the name of the data type of the *SPACE_FooTypeSupport*.

Detailed Description

This operation returns the default name of the data type associated with the *SPACE_FooTypeSupport*. The default name is derived from the type name as specified in the IDL definition. It is composed of the scope names and the type name, each separated by “:”, in order of lower scope level to deeper scope level followed by the type name.

3.3.6.8 SPACE_FooTypeSupport_register_type

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooTypeSupport_register_type
        (SPACE_FooTypeSupport _this,
         DDS_DomainParticipant domain,
         DDS_string type_name);
```

Description

This operation registers a new data type name to a DDS_DomainParticipant.

Parameters

in SPACE_FooTypeSupport _this - the SPACE_FooTypeSupport object on which the operation is operated.

in DDS_DomainParticipant domain - a pointer to a DDS_DomainParticipant object to which the new data type is registered.

in DDS_string type_name - a local alias of the new data type to be registered.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation registers a new data type name to a DDS_DomainParticipant. This operation informs the Data Distribution Service, in order to allow it to manage the new registered data type. This operation also informs the Data Distribution Service about the key definition, which allows the Data Distribution Service to distinguish different instances of the same data type.

Precondition

A *type_name* cannot be registered with two different *<NameSpace>_<type>TypeSupport* classes (this means of a different data type) with the same DDS_DomainParticipant. When the operation is called on the same DDS_DomainParticipant with the same *type_name* for a different *<NameSpace>_<type>TypeSupport* class, the operation returns DDS_RETCODE_PRECONDITION_NOT_MET. However, it is possible to register the same *<NameSpace>_<type>TypeSupport* classes with the same

DDS_DomainParticipant and the same or different type_name multiple times. All registrations return DDS_RETCODE_OK, but any subsequent registrations with the same type_name are ignored.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the SPACE_FooTypeSupport class is registered with the new data type name to the DDS_DomainParticipant or the SPACE_FooTypeSupport class was already registered.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - one or both parameters is the DDS_OBJECT_NIL pointer or the parameter type_name has zero length.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - this type_name is already registered with this DDS_DomainParticipant for a different <NameSpace>_<type>TypeSupport class.

3.4 Publication Module

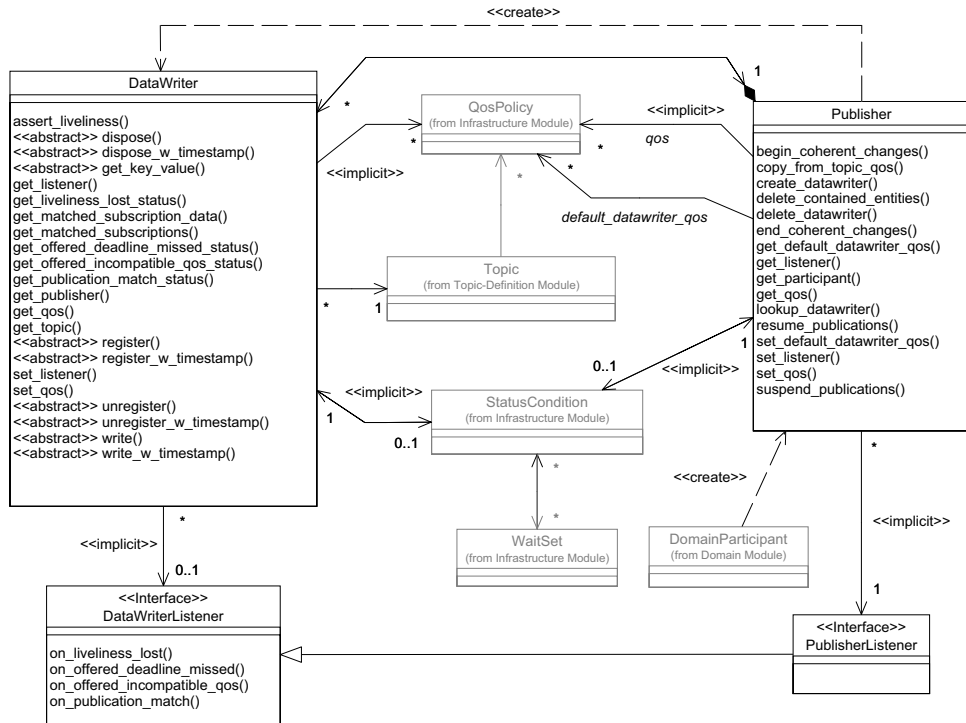


Figure 18 The DCPS Publication Module’s Class Model

This module contains the following classes:

- DDS_Publisher
- Publication type specific classes
- DDS_PublisherListener (interface)
- DDS_DataWriterListener (interface).

The paragraph “Publication type specific classes” contains the generic class and the generated data type specific classes. For each data type, a data type specific class `<NameSpace>_<type>DataWriter` is generated (based on IDL) by calling the pre-processor.

For instance, for the fictional data type `Foo` (this also applies to other types), defined in the module `SPACE`; “Publication type specific classes” contains the following classes:

- DDS_DataWriter (abstract)
- SPACE_FooDataWriter.

A `DDS_Publisher` is an object responsible for data distribution. It may publish data of different data types. A `DDS_DataWriter` acts as a typed accessor to a `DDS_Publisher`. The `DDS_DataWriter` is the object the application must use to communicate the existence and value of data-objects of a given data type to a `DDS_Publisher`. When data-object values have been communicated to the `DDS_Publisher` through the appropriate `DDS_DataWriter`, it is the `DDS_Publisher`'s responsibility to perform the distribution. The `DDS_Publisher` will do this according to its own `DDS_PublisherQos`, and the `DDS_DataWriterQos` attached to the corresponding `DDS_DataWriter`. A publication is defined by the association of a `DDS_DataWriter` to a `DDS_Publisher`. This association expresses the intent of the application to publish the data described by the `DDS_DataWriter` in the context provided by the `DDS_Publisher`.

3.4.1 Class `DDS_Publisher`

The `DDS_Publisher` acts on behalf of one or more `DDS_DataWriter` objects that belong to it. When it is informed of a change to the data associated with one of its `DDS_DataWriter` objects, it decides when it is appropriate to actually process the sample-update message. In making this decision, it considers the `DDS_PublisherQos` and the `DDS_DataWriterQos`.

The interface description of this class is as follows:

```

/*
 * interface DDS_Publisher
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *   DDS_Publisher_get_statuscondition
 *   (DDS_Publisher _this);
 */
/* DDS_StatusMask
 *   DDS_Publisher_get_status_changes
 *   (DDS_Publisher _this);
 */
/* DDS_ReturnCode_t
 *   DDS_Publisher_enable
 *   (DDS_Publisher _this);
 */
/*
 * implemented API operations
 */
DDS_DataWriter
DDS_Publisher_create_datawriter
(DDS_Publisher _this,
```

```

        const DDS_Topic a_topic,
        const DDS_DataWriterQos *qos,
        const struct DDS_DataWriterListener *a_listener,
        const DDS_StatusMask mask);

DDS_ReturnCode_t
    DDS_Publisher_delete_datawriter
        (DDS_Publisher _this,
         const DDS_DataWriter a_datawriter);

DDS_DataWriter
    DDS_Publisher_lookup_datawriter
        (DDS_Publisher _this,
         const DDS_char *topic_name);
DDS_ReturnCode_t
    DDS_Publisher_delete_contained_entities
        (DDS_Publisher _this);

DDS_ReturnCode_t
    DDS_Publisher_set_qos
        (DDS_Publisher _this,
         const DDS_PublisherQos *qos);
DDS_ReturnCode_t
    DDS_Publisher_get_qos
        (DDS_Publisher _this,
         DDS_PublisherQos *qos);
DDS_ReturnCode_t
    DDS_Publisher_set_listener
        (DDS_Publisher _this,
         const struct DDS_PublisherListener *a_listener,
         const DDS_StatusMask mask);
struct DDS_PublisherListener
    DDS_Publisher_get_listener
        (DDS_Publisher _this);
DDS_ReturnCode_t
    DDS_Publisher_suspend_publications
        (DDS_Publisher _this);

DDS_ReturnCode_t
    DDS_Publisher_resume_publications
        (DDS_Publisher _this);

DDS_ReturnCode_t
    DDS_Publisher_begin_coherent_changes
        (DDS_Publisher _this);

DDS_ReturnCode_t
    DDS_Publisher_end_coherent_changes
        (DDS_Publisher _this);

```



```

DDS_ReturnCode_t
    DDS_Publisher_wait_for_acknowledgments
        (DDS_Publisher _this,
         const DDS_Duration_t *max_wait);

DDS_DomainParticipant
    DDS_Publisher_get_participant
        (DDS_Publisher _this);

DDS_ReturnCode_t
    DDS_Publisher_set_default_datawriter_qos
        (DDS_Publisher _this,
         const DDS_DataWriterQos *qos);

DDS_ReturnCode_t
    DDS_Publisher_get_default_datawriter_qos
        (DDS_Publisher _this,
         DDS_DataWriterQos *qos);

DDS_ReturnCode_t
    DDS_Publisher_copy_from_topic_qos
        (DDS_Publisher _this,
         DDS_DataWriterQos *a_datawriter_qos,
         const DDS_TopicQos *a_topic_qos);

```

The next paragraphs describe the usage of all `DDS_Publisher` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.4.1.1 DDS_Publisher_begin_coherent_changes

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_begin_coherent_changes
        (DDS_Publisher _this);

```

Description

This operation requests that the application will begin a ‘coherent set’ of modifications using `DDS_DataWriter` objects attached to this `DDS_Publisher`. The ‘coherent set’ will be completed by a matching call to `DDS_Publisher_end_coherent_changes`.

Parameters

in `DDS_Publisher _this` - the `DDS_Publisher` object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation requests that the application will begin a ‘coherent set’ of modifications using *DDS_DataWriter* objects attached to this *DDS_Publisher*. The ‘coherent set’ will be completed by a matching call to *DDS_Publisher_end_coherent_changes*.

A ‘coherent set’ is a set of modifications that must be propagated in such a way that they are interpreted at the receivers’ side as a consistent set of modifications; that is, the receiver will only be able to access the data after all the modifications in the set are available at the receiver end.

A precondition for making coherent changes is that the *PresentationQos* of the *DDS_Publisher* has its *coherent_access* attribute set to *TRUE*. If this is not the case, the *Publisher* will not accept any coherent start requests and return *DDS_RETCODE_PRECONDITION_NOT_MET*.

A connectivity change may occur in the middle of a set of coherent changes; for example, the set of partitions used by the *DDS_Publisher* or one of its connected *DDS_Subscribers* may change, a late-joining *DDS_DataReader* may appear on the network, or a communication failure may occur. In the event that such a change prevents an entity from receiving the entire set of coherent changes, that entity must behave as if it had received none of the set.

These calls can be nested. In that case, the coherent set terminates only with the last call to *DDS_Publisher_end_coherent_changes*.

The support for ‘coherent changes’ enables a publishing application to change the value of several data-instances that could belong to the same or different topics and have those changes be seen ‘atomically’ by the readers. This is useful in cases where the values are inter-related (for example, if there are two data-instances representing the ‘altitude’ and ‘velocity vector’ of the same aircraft and both are changed, it may be useful to communicate those values in a way the reader can see both together; otherwise, it may e.g., erroneously interpret that the aircraft is on a collision course).

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a new coherent change has successfully been started.
- *DDS_RETCODE_ERROR* - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter passed to the operation is NULL, or is not pointing to any valid object.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Publisher has already been deleted.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the DDS_Publisher is not able to handle coherent changes because its PresentationQos has not set *coherent_access* to TRUE.

3.4.1.2 DDS_Publisher_copy_from_topic_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_copy_from_topic_qos
        (DDS_Publisher _this,
         DDS_DataWriterQos *a_datawriter_qos,
         const DDS_TopicQos *a_topic_qos);
```

Description

This operation will copy policies in *a_topic_qos* to the corresponding policies in *a_datawriter_qos*.

Parameters

in DDS_Publisher _this - the DDS_Publisher object on which the operation is operated.

*inout DDS_DataWriterQos *a_datawriter_qos* - the destination DDS_DataWriterQos struct to which the QosPolicy settings should be copied.

*in const DDS_TopicQos *a_topic_qos* - the source DDS_TopicQos struct, which should be copied.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation will copy the `QosPolicy` settings in `a_topic_qos` to the corresponding `QosPolicy` settings in `a_datawriter_qos` (replacing the values in `a_datawriter_qos`, if present). This will only apply to the common `QosPolicy` settings in each `<DDS_Entity>Qos`.

This is a “convenience” operation, useful in combination with the operations `DDS_Publisher_get_default_datawriter_qos` and `DDS_Topic_get_qos`. The operation `DDS_Publisher_copy_from_topic_qos` can be used to merge the `DDS_DataWriter` default `QosPolicy` settings with the corresponding ones on the `DDS_TopicQos`. The resulting `DDS_DataWriterQos` can then be used to create a new `DDS_DataWriter`, or set its `DDS_DataWriterQos`.

This operation does not check the resulting `a_datawriter_qos` for consistency. This is because the “merged” `a_datawriter_qos` may not be the final one, as the application can still modify some `QosPolicy` settings prior to applying the `DDS_DataWriterQos` to the `DDS_DataWriter`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `QosPolicy` settings have successfully been copied from the `DDS_TopicQos` to the `DDS_DataWriterQos`.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Publisher` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.4.1.3 DDS_Publisher_create_datawriter

Synopsis

```
#include <dds_dcps.h>
DDS_DataWriter
    DDS_Publisher_create_datawriter
        (DDS_Publisher _this,
         const DDS_Topic a_topic,
         const DDS_DataWriterQos *qos,
         const struct DDS_DataWriterListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation creates a `DDS_DataWriter` with the desired `DDS_DataWriterQos`, for the desired `DDS_Topic` and attaches the optionally specified `DDS_DataWriterListener` to it.

Parameters

in `DDS_Publisher _this` - the `DDS_Publisher` object on which the operation is operated.

in `const DDS_Topic a_topic` - a pointer to the topic for which the `DDS_DataWriter` is created.

*in `const DDS_DataWriterQos *qos`* - the `DDS_DataWriterQos` for the new `DDS_DataWriter`. In case these settings are not self consistent, no `DDS_DataWriter` is created.

*in `const struct DDS_DataWriterListener *a_listener`* - a pointer to the `DDS_DataWriterListener` instance which will be attached to the new `DDS_DataWriter`. It is permitted to use `DDS_OBJECT_NIL` as the value of the listener: this behaves as a `DDS_DataWriterListener` whose operations perform no action.

in `const DDS_StatusMask mask` - a bit-mask in which each bit enables the invocation of the `DDS_DataWriterListener` for a certain status.

Return Value

`DDS_DataWriter` - Return value is a pointer to the newly created `DDS_DataWriter`. In case of an error, the `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a `DDS_DataWriter` with the desired `DDS_DataWriterQos`, for the desired `DDS_Topic` and attaches the optionally specified `DDS_DataWriterListener` to it. The returned `DDS_DataWriter` is attached (and belongs) to the `DDS_Publisher` on which this operation is being called. To delete the `DDS_DataWriter` the operation `DDS_Publisher_delete_datawriter` or `DDS_Publisher_delete_contained_entities` must be used. If no write rights are defined for the specific topic then the creation of the `DataWriter` will fail.

Application Data Type

The `DDS_DataWriter` returned by this operation is an object of a derived class, specific to the data type associated with the `DDS_Topic`. For each application-defined data type `<type>` there is a class

<NameSpace>_<type>DataWriter generated by calling the pre-processor. This data type specific class extends DDS_DataWriter and contains the operations to write data of data type <type>.

QosPolicy

The possible application pattern to construct the DDS_DataWriterQos for the DDS_DataWriter is to:

- Retrieve the QosPolicy settings on the associated DDS_Topic by means of the get_qos operation on the DDS_Topic.
- Retrieve the default DDS_DataWriterQos by means of the DDS_Publisher_get_default_datawriter_qos operation on the DDS_Publisher
- Combine those two lists of QosPolicy settings and selectively modify QosPolicy settings as desired
- Use the resulting DDS_DataWriterQos to construct the DDS_DataWriter.
- In case the specified QosPolicy settings are not consistent, no DDS_DataWriter is created and the DDS_OBJECT_NIL pointer is returned.

Default QoS

The constant DDS_DATAWRITER_QOS_DEFAULT can be used as parameter qos to create a DDS_DataWriter with the default DDS_DataWriterQos as set in the DDS_Publisher. The effect of using DDS_DATAWRITER_QOS_DEFAULT is the same as calling the operation DDS_Publisher_get_default_datawriter_qos and using the resulting DDS_DataWriterQos to create the DDS_DataWriter.

The special DDS_DATAWRITER_QOS_USE_TOPIC_QOS can be used to create a DDS_DataWriter with a combination of the default DDS_DataWriterQos and the DDS_TopicQos. The effect of using DDS_DATAWRITER_QOS_USE_TOPIC_QOS is the same as calling the operation DDS_Publisher_get_default_datawriter_qos and retrieving the DDS_TopicQos (by means of the operation DDS_Topic_get_qos) and then combining these two QosPolicy settings using the operation DDS_Publisher_copy_from_topic_qos, whereby any common policy that is set on the DDS_TopicQos “overrides” the corresponding policy on the default DDS_DataWriterQos. The resulting DDS_DataWriterQos is then applied to create the DDS_DataWriter.

Communication Status

For each communication status, the StatusChangedFlag flag is initially set to FALSE. It becomes TRUE whenever that communication status changes. For each communication status activated in the mask, the associated

DDS_DataWriterListener operation is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The fact that the status is reset prior to calling the listener means that if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset.

The following statuses are applicable to the DDS_DataWriterListener:

```
DDS_OFFERED_DEADLINE_MISSED_STATUS
DDS_OFFERED_INCOMPATIBLE_QOS_STATUS
DDS_LIVELINESS_LOST_STATUS
DDS_PUBLICATION_MATCHED_STATUS.
```



Be aware that the DDS_PUBLICATION_MATCHED_STATUS is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return DDS_OBJECT_NIL.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant DDS_STATUS_MASK_NONE can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant STATUS_MASK_ANY_V1_2 can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

In case a communication status is not activated in the mask of the DDS_DataWriterListener, the DDS_PublisherListener of the containing DDS_Publisher is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the DDS_PublisherListener of the containing DDS_Publisher and a DDS_DataWriter specific behaviour when needed. In case the communication status is not activated in the mask of the DDS_PublisherListener as well, the communication status will be propagated to the DDS_DomainParticipantListener of the containing DDS_DomainParticipant. In case the DDS_DomainParticipantListener is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

3.4.1.4 DDS_Publisher_delete_contained_entities

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_delete_contained_entities
        (DDS_Publisher _this);
```

Description

This operation deletes all the `DDS_DataWriter` objects that were created by means of one of the `DDS_Publisher_create_datawriter` operations on the `DDS_Publisher`.

Parameters

in `DDS_Publisher _this` - the `DDS_Publisher` object on which the operation is operated.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation deletes all the `DDS_DataWriter` objects that were created by means of one of the `DDS_Publisher_create_datawriter` operations on the `DDS_Publisher`. In other words, it deletes all contained `DDS_DataWriter` objects.



NOTE: The operation will return `DDS_PRECONDITION_NOT_MET` if the any of the contained entities is in a state where it cannot be deleted. In such cases, the operation does not roll back any entity deletions performed prior to the detection of the problem.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the contained `DDS_Entity` objects are deleted and the application may delete the `DDS_Publisher`.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.

- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Publisher* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one or more of the contained entities are in a state where they cannot be deleted.

3.4.1.5 DDS_Publisher_delete_datawriter

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_delete_datawriter
        (DDS_Publisher _this,
         const DDS_DataWriter a_datawriter);
```

Description

This operation deletes a *DDS_DataWriter* that belongs to the *DDS_Publisher*.

Parameters

in DDS_Publisher _this - the *DDS_Publisher* object on which the operation -is operated.

in const DDS_DataWriter a_datawriter - a pointer to the *DDS_DataWriter*, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_PRECONDITION_NOT_MET*.

Detailed Description

This operation deletes a *DDS_DataWriter* that belongs to the *DDS_Publisher*. When the operation is called on a different *DDS_Publisher*, as used when the *DDS_DataWriter* was created, the operation has no effect and returns *DDS_RETCODE_PRECONDITION_NOT_MET*. The deletion of the *DDS_DataWriter* will automatically unregister all instances. Depending on the settings of *DDS_WriterDataLifecycleQosPolicy*, the deletion of the *DDS_DataWriter* may also dispose of all instances.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_DataWriter` is deleted.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `a_datawriter` is not a valid `DDS_DataWriter`.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Publisher` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the operation is called on a different `DDS_Publisher`, as used when the `DDS_DataWriter` was created.

3.4.1.6 `DDS_Publisher_enable` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_enable
        (DDS_Publisher _this);
```

3.4.1.7 `DDS_Publisher_end_coherent_changes`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_end_coherent_changes
        (DDS_Publisher _this);
```

Description

This operation terminates the ‘coherent set’ initiated by the matching call to `DDS_Publisher_begin_coherent_changes`.

Parameters

in `DDS_Publisher _this` - the `DDS_Publisher` object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation terminates the ‘coherent set’ initiated by the matching call to *DDS_Publisher_begin_coherent_changes*. If there is no matching call to *DDS_Publisher_begin_coherent_changes*, the operation will return the error *DDS_PRECONDITION_NOT_MET*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the coherent change has successfully been closed.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter passed to the operation is NULL, or is not pointing to any valid object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Publisher* has already been deleted.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - there is no matching *DDS_Publisher_begin_coherent_changes* call that can be closed.

3.4.1.8 DDS_Publisher_get_default_datawriter_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
  DDS_Publisher_get_default_datawriter_qos
    (DDS_Publisher _this,
     DDS_DataWriterQos *qos);
```

Description

This operation gets the default *DDS_DataWriterQos* of the *DDS_Publisher*.

Parameters

in DDS_Publisher _this - the *DDS_Publisher* object on which the operation is operated.

*inout DDS_DataWriterQos *qos* - a pointer to the DDS_DataWriterQos struct (provided by the application) in which the default DDS_DataWriterQos for the DDS_DataWriter is written.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation gets the default DDS_DataWriterQos of the DDS_Publisher (that is the struct with the QosPolicy settings) which is used for newly created DDS_DataWriter objects, in case the constant DDS_DATAWRITER_QOS_DEFAULT is used. The default DDS_DataWriterQos is only used when the constant is supplied as parameter qos to specify the DDS_DataWriterQos in the DDS_Publisher_create_datawriter operation. The application must provide the DDS_DataWriterQos struct in which the QosPolicy settings can be stored and pass the qos pointer to the operation. The operation writes the default DDS_DataWriterQos to the struct pointed to by qos. Any settings in the struct are overwritten.

The values retrieved by this operation match the set of values specified on the last successful call to DDS_Publisher_set_default_datawriter_qos, or, if the call was never made, the default values as specified for each QosPolicy setting as defined in Table 5 on page 59.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the default DDS_DataWriter QosPolicy settings of this DDS_Publisher have successfully been copied into the specified DDS_DataWriterQos parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Publisher has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.1.9 DDS_Publisher_get_listener

Synopsis

```
#include <dds_dcps.h>
struct DDS_PublisherListener
    DDS_Publisher_get_listener
        (DDS_Publisher _this);
```

Description

This operation allows access to a DDS_PublisherListener.

Parameters

in DDS_Publisher _this - the DDS_Publisher object on which the operation is operated.

Return Value

struct DDS_PublisherListener - a pointer to the DDS_PublisherListener attached to the DDS_Publisher.

Detailed Description

This operation allows access to a DDS_PublisherListener attached to the DDS_Publisher. When no DDS_PublisherListener was attached to the DDS_Publisher, the DDS_OBJECT_NIL pointer is returned.

3.4.1.10 DDS_Publisher_get_participant

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_Publisher_get_participant
        (DDS_Publisher _this);
```

Description

This operation returns the DDS_DomainParticipant associated with the DDS_Publisher or the DDS_OBJECT_NIL pointer.

Parameters

in DDS_Publisher _this - the DDS_Publisher object on which the operation is operated.

Return Value

DDS_DomainParticipant - a pointer to the *DDS_DomainParticipant* associated with the *DDS_Publisher* or the *DDS_OBJECT_NIL* pointer.

Detailed Description

This operation returns the *DDS_DomainParticipant* associated with the *DDS_Publisher*. Note that there is exactly one *DDS_DomainParticipant* associated with each *DDS_Publisher*. When the *DDS_Publisher* was already deleted (there is no associated *DDS_DomainParticipant* any more), the *DDS_OBJECT_NIL* pointer is returned.

3.4.1.11 DDS_Publisher_get_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_get_qos
        (DDS_Publisher _this,
         DDS_PublisherQos *qos);
```

Description

This operation allows access to the existing set of QoS policies for a *DDS_Publisher*.

Parameters

in DDS_Publisher _this - the *DDS_Publisher* object on which the operation is operated.

*inout DDS_PublisherQos *qos* - a pointer to the destination *DDS_PublisherQos* struct in which the *QosPolicy* settings will be copied.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation allows access to the existing set of QoS policies of a *DDS_Publisher* on which this operation is used. This *DDS_PublisherQos* is stored at the location pointed to by the *qos* parameter.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the existing set of QoS policy values applied to this *DDS_Publisher* has successfully been copied into the specified *DDS_PublisherQos* parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Publisher* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.1.12 DDS_Publisher_get_status_changes (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
    DDS_Publisher_get_status_changes
        (DDS_Publisher _this);
```

3.4.1.13 DDS_Publisher_get_statuscondition (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
    DDS_Publisher_get_statuscondition
        (DDS_Publisher _this);
```

3.4.1.14 DDS_Publisher_lookup_datawriter**Synopsis**

```
#include <dds_dcps.h>
DDS_DataWriter
    DDS_Publisher_lookup_datawriter
        (DDS_Publisher _this,
         const DDS_char *topic_name);
```

Description

This operation returns a previously created `DDS_DataWriter` belonging to the `DDS_Publisher` which is attached to a `DDS_Topic` with the matching `topic_name`.

Parameters

in `DDS_Publisher _this` - the `DDS_Publisher` object on which the operation is operated.

in `const DDS_char *topic_name` - the name of the `DDS_Topic`, which is attached to the `DDS_DataWriter` to look for.

Return Value

`DDS_DataWriter` - Return value is a pointer to the `DDS_DataWriter` found. When no such `DDS_DataWriter` is found, the `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation returns a previously created `DDS_DataWriter` belonging to the `DDS_Publisher` which is attached to a `DDS_Topic` with the matching `topic_name`. When multiple `DDS_DataWriter` objects (which satisfy the same condition) exist, this operation will return one of them. It is not specified which one.

3.4.1.15 DDS_Publisher_resume_publications**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_resume_publications
        (DDS_Publisher _this);
```

Description

This operation resumes a previously suspended publication.

Parameters

in `DDS_Publisher _this` - the `DDS_Publisher` object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

If the *DDS_Publisher* is suspended, this operation will resume the publication of all *DDS_DataWriter* objects contained by this *DDS_Publisher*. All data held in the history buffer of the *DDS_DataWriter*'s is actively published to the consumers. When the operation returns, all *DDS_DataWriter*'s have resumed the publication of suspended updates.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the *DDS_Publisher* object has been resumed.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Publisher* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *DDS_Publisher* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the *DDS_Publisher* is not suspended.

3.4.1.16 *DDS_Publisher_set_default_datawriter_qos*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_set_default_datawriter_qos
        (DDS_Publisher _this,
         const DDS_DataWriterQos *qos);
```

Description

This operation sets the default *DDS_DataWriterQos* of the *DDS_Publisher*.

Parameters

in DDS_Publisher_this - the DDS_Publisher object on which the operation is operated.

*in const DDS_DataWriterQos *qos* - the DDS_DataWriterQos struct, which contains the new default DDS_DataWriterQos for the newly created DDS_DataWriters.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_INCONSISTENT_POLICY.

Detailed Description

This operation sets the default DDS_DataWriterQos of the DDS_Publisher (that is the struct with the QosPolicy settings) which is used for newly created DDS_DataWriter objects, in case the constant DDS_DATAWRITER_QOS_DEFAULT is used. The default DDS_DataWriterQos is only used when the constant is supplied as parameter qos to specify the DDS_DataWriterQos in the DDS_Publisher_create_datawriter operation.

The DDS_Publisher_set_default_datawriter_qos operation checks if the DDS_DataWriterQos is self consistent. If it is not, the operation has no effect and returns DDS_RETCODE_INCONSISTENT_POLICY.

The values set by this operation are returned by DDS_Publisher_get_default_datawriter_qos.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new default DDS_DataWriterQos is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter qos is not a valid DDS_DataWriterQos. It contains a QosPolicy setting with an invalid DDS_Duration_t value, an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- *DDS_RETCODE_UNSUPPORTED* - one or more of the selected QosPolicy values are currently not supported by OpenSplice.

- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Publisher* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_INCONSISTENT_POLICY* - the parameter *qos* contains conflicting *QosPolicy* settings, e.g. a history depth that is higher than the specified resource limits.

3.4.1.17 DDS_Publisher_set_listener

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_set_listener
        (DDS_Publisher _this,
         const struct DDS_PublisherListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation attaches a *DDS_PublisherListener* to the *DDS_Publisher*.

Parameters

in DDS_Publisher _this - the *DDS_Publisher* object on which the operation is operated.

*in const struct DDS_PublisherListener *a_listener* - a pointer to the *DDS_PublisherListener* instance, which will be attached to the *DDS_Publisher*.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the *DDS_PublisherListener* for a certain status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation attaches a *DDS_PublisherListener* to the *DDS_Publisher*. Only one *DDS_PublisherListener* can be attached to each *DDS_Publisher*. If a *DDS_PublisherListener* was already attached, the operation will replace it

with the new one. When `a_listener` is the `DDS_OBJECT_NIL` pointer, it represents a listener that is treated as a NOOP¹ for all statuses activated in the bitmask.

Communication Status

For each communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever that communication status changes. For each communication status activated in the `mask`, the associated `DDS_PublisherListener` operation is invoked and the communication status is reset to `FALSE`, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset. An exception to this rule is the `DDS_OBJECT_NIL` listener, which does not reset the communication statuses for which it is invoked.

The following statuses are applicable to the `DDS_PublisherListener`:

- `DDS_OFFERED_DEADLINE_MISSED_STATUS` (propagated)
- `DDS_OFFERED_INCOMPATIBLE_QOS_STATUS` (propagated)
- `DDS_LIVELINESS_LOST_STATUS` (propagated)
- `DDS_PUBLICATION_MATCHED_STATUS` (propagated).



Be aware that the `DDS_PUBLICATION_MATCHED_STATUS` is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

The Data Distribution Service will trigger the most specific and relevant `Listener`. In other words, in case a communication status is also activated on the `DDS_DataWriterListener` of a contained `DDS_DataWriter`, the

1. Short for **No-Operation**, an instruction that performs nothing at all.

DDS_DataWriterListener on that contained DDS_DataWriter is invoked instead of the DDS_PublisherListener. This means that a status change on a contained DDS_DataWriter only invokes the DDS_PublisherListener if the contained DDS_DataWriter itself does not handle the trigger event generated by the status change.

In case a status is not activated in the mask of the DDS_PublisherListener, the DDS_DomainParticipantListener of the containing DDS_DomainParticipant is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the DDS_DomainParticipantListener of the containing DDS_DomainParticipant and a DDS_Publisher specific behaviour when needed. In case the DDS_DomainParticipantListener is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_PublisherListener is attached.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_UNSUPPORTED* - a status was selected that cannot be supported because the infrastructure does not maintain the required connectivity information.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Publisher has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.1.18 DDS_Publisher_set_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_set_qos
        (DDS_Publisher _this,
         const DDS_PublisherQos *qos);
```

Description

This operation replaces the existing set of QosPolicy settings for a DDS_Publisher.

Parameters

in DDS_Publisher _this - the DDS_Publisher object on which the operation is operated.

*in const DDS_PublisherQos *qos* - contains the new set of QosPolicy settings for the DDS_Publisher.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_IMMUTABLE_POLICY or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation replaces the existing set of QosPolicy settings for a DDS_Publisher. The parameter *qos* contains the QosPolicy settings which is checked for self-consistency and mutability. When the application tries to change a QosPolicy setting for an enabled DDS_Publisher, which can only be set before the DDS_Publisher is enabled, the operation will fail and a DDS_RETCODE_IMMUTABLE_POLICY is returned. In other words, the application must provide the currently set QosPolicy settings in case of the immutable QosPolicy settings. Only the mutable QosPolicy settings can be changed. When *qos* contains conflicting QosPolicy settings (not self-consistent), the operation will fail and a DDS_RETCODE_INCONSISTENT_POLICY is returned.

The set of QosPolicy settings specified by the *qos* parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned DDS_RETCODE_OK). If one or more of the partitions in the QoS structure have insufficient access rights configured then the *set_qos* function will fail with a DDS_RETCODE_PRECONDITION_NOT_MET error code.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new DDS_PublisherQos is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *qos* is not a valid DDS_PublisherQos. It contains a QosPolicy setting with an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.

- *DDS_RETCODE_UNSUPPORTED* - one or more of the selected *QoSPolicy* values are currently not supported by OpenSplice.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Publisher* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_IMMUTABLE_POLICY* - the parameter *qos* contains an immutable *QoSPolicy* setting with a different value than set during enabling of the *DDS_Publisher*.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - returned when insufficient access rights exist for the partition(s) listed in the *QoS* structure.

3.4.1.19 DDS_Publisher_suspend_publications

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_suspend_publications
        (DDS_Publisher _this);
```

Description

This operation will suspend the dissemination of the publications by all contained *DataWriter* objects.

Parameters

in *DDS_Publisher _this* - the *DDS_Publisher* object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_NOT_ENABLED*.

Detailed Description

This operation suspends the publication of all *DDS_DataWriter* objects contained by this *DDS_Publisher*. The data written, disposed or unregistered by a *DDS_DataWriter* is stored in the history buffer of the *DDS_DataWriter* and therefore, depending on its *QoS* settings, the following operations may block (see the operation descriptions for more information):

- *DDS_DataWriter_dispose*

- `DDS_DataWriter_dispose_w_timestamp`
- `DDS_DataWriter_write`
- `DDS_DataWriter_write_w_timestamp`
- `DDS_DataWriter_writedispose`
- `DDS_DataWriter_writedispose_w_timestamp`
- `DDS_DataWriter_unregister_instance`
- `DDS_DataWriter_unregister_instance_w_timestamp`

Subsequent calls to the `DDS_Publisher_suspend_publications` operation have no effect. When the `DDS_Publisher` is deleted before `DDS_Publisher_resume_publications` is called, all suspended updates are discarded.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_Publisher` has been suspended.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Publisher` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_Publisher` is not enabled.

3.4.1.20 `DDS_Publisher_wait_for_acknowledgments`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Publisher_wait_for_acknowledgments
        (DDS_Publisher _this,
         const DDS_Duration_t *max_wait);
```

Description

This operation blocks the calling thread until either all data written by all contained `DataWriters` is acknowledged by the local infrastructure, or until the duration specified by `max_wait` parameter elapses, whichever happens first.

Parameters

in DDS_Publisher _this - the DDS_Publisher object on which the operation is operated.

*in const DDS_Duration_t *max_wait* - the maximum duration to block for the DDS_Publisher_wait_for_acknowledgments, after which the application thread is unblocked. The special constant DDS_DURATION_INFINITE can be used when the maximum waiting time does not need to be bounded.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED or DDS_RETCODE_TIMEOUT.

Detailed Description

This operation blocks the calling thread until either all data written by all contained DataWriters is acknowledged by the local infrastructure, or until the duration specified by *max_wait* parameter elapses, whichever happens first.

Data is acknowledged by the local infrastructure when it does not need to be stored in its DataWriter's local history. When a locally-connected subscription (including the networking service) has no more resources to store incoming samples it will start to reject these samples, resulting in their source DataWriters to store them temporarily in their own local history to be retransmitted at a later moment in time. In such scenarios, the *DDS_Publisher_wait_for_acknowledgments* operation will block until all contained DataWriters have retransmitted their entire history, which is therefore effectively empty, or until the *max_wait* timeout expires, whichever happens first. In the first case the operation will return DDS_RETCODE_OK, in the latter it will return DDS_RETCODE_TIMEOUT.



Be aware that in case the operation returns DDS_RETCODE_OK, the data has only been acknowledged by the local infrastructure: it does not mean all remote subscriptions have already received the data. However, delivering the data to remote nodes is then the sole responsibility of the networking service: even when the publishing application would terminate, all data that has not yet been received may be considered 'on-route' and will therefore eventually arrive (unless the networking service itself crashes). In contrast, if a DataWriter would still have data in its local history buffer when it terminates, this data is considered 'lost'.

This operation is intended to be used only if one or more of the contained `DataWriters` has its `DDS_ReliabilityQosPolicyKind` set to `DDS_RELIABLE_RELIABILITY_QOS`. Otherwise the operation will return immediately with `DDS_RETCODE_OK`, since best-effort `DataWriters` will never store rejected samples in their local history: they will just drop them and continue business as usual.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the data of all contained `DataWriters` has been acknowledged by the local infrastructure.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Publisher` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_Publisher` is not enabled.
- `DDS_RETCODE_TIMEOUT` - not all data is acknowledged before `max_wait` elapsed.

3.4.2 Publication Type Specific Classes

This paragraph describes the generic `DDS_DataWriter` class and the derived application type specific `<NameSpace>_<type>DataWriter` classes which together implement the application publication interface. For each application type, used as `DDS_Topic` data type, the pre-processor generates a `<NameSpace>_<type>DataWriter` class from an IDL type description. The `SPACE_FooDataWriter` class that would be generated by the pre-processor for a fictional type `Foo` (defined in the module `SPACE`) describes the `<NameSpace>_<type>DataWriter` classes.

3.4.2.1 Class `DDS_DataWriter` (abstract)

`DDS_DataWriter` allows the application to set the value of the sample to be published under a given `DDS_Topic`.

A `DDS_DataWriter` is attached to exactly one `DDS_Publisher` which acts as a factory for it.

A `DDS_DataWriter` is bound to exactly one `DDS_Topic` and therefore to exactly one data type. The `DDS_Topic` must exist prior to the `DDS_DataWriter`'s creation.

`DDS_DataWriter` is an abstract class. It must be specialized for each particular application data type. For a fictional application data type `Foo` (defined in the module `SPACE`) the specialized class would be `SPACE_FooDataWriter`.

The interface description of this class is as follows:

```

/*
 * interface DDS_DataWriter
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *   DDS_DataWriter_get_statuscondition
 *   (DDS_DataWriter _this);
 */
/* DDS_StatusMask
 *   DDS_DataWriter_get_status_changes
 *   (DDS_DataWriter _this);
 */
/* DDS_ReturnCode_t
 *   DDS_DataWriter_enable
 *   (DDS_DataWriter _this);
 */
/*
 * abstract operations
 * (implemented in the data type specific DDS_DataWriter)
 */
/* DDS_InstanceHandle_t
 *   DDS_DataWriter_register_instance
 *   (DDS_DataWriter _this);
 *   const <data> *instance_data);
 */
/* DDS_InstanceHandle_t
 *   DDS_DataWriter_register_instance_w_timestamp
 *   (DDS_DataWriter _this);
 *   const <data> *instance_data,
 *   const DDS_Time_t *source_timestamp);
 */
/* DDS_ReturnCode_t
 *   DDS_DataWriter_unregister_instance
 *   (DDS_DataWriter _this);
 *   const <data> *instance_data,
 *   const DDS_InstanceHandle_t handle);
 */
/* DDS_ReturnCode_t

```

```

*     DDS_DataWriter_unregister_instance_w_timestamp
*     (DDS_DataWriter _this);
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t handle,
*     const DDS_Time_t *source_timestamp);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_write
*     (DDS_DataWriter _this);
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t handle);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_write_w_timestamp
*     (DDS_DataWriter _this);
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t handle,
*     const DDS_Time_t *source_timestamp);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_dispose
*     (DDS_DataWriter _this);
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t instance_handle);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_dispose_w_timestamp
*     (DDS_DataWriter _this);
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t instance_handle,
*     const DDS_Time_t *source_timestamp);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_writedispose
*     (DDS_DataWriter _this,
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t instance_handle);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_writedispose_w_timestamp
*     (DDS_DataWriter _this,
*     const <data> *instance_data,
*     const DDS_InstanceHandle_t instance_handle,
*     const DDS_Time_t *source_timestamp);
*/
/* DDS_ReturnCode_t
*     DDS_DataWriter_get_key_value
*     (DDS_DataWriter _this);
*     <data> *key_holder,

```

```

*          const DDS_InstanceHandle_t handle);
*/
/* DDS_InstanceHandle_t
*   DDS_DataWriter_lookup_instance
*/   (DDS_DataWriter _this,
*       <data> *instance_data);
/*
/*
* implemented API operations
*/
    DDS_ReturnCode_t
        DDS_DataWriter_set_qos
            (DDS_DataWriter this,
             const DDS_DataWriterQos *qos);

    DDS_ReturnCode_t
        DDS_DataWriter_get_qos
            (DDS_DataWriter this,
             DDS_DataWriterQos *qos);

    DDS_ReturnCode_t
        DDS_DataWriter_set_listener
            (DDS_DataWriter this,
             const struct DDS_DataWriterListener *a_listener,
             const DDS_StatusMask mask);

    struct DDS_DataWriterListener
        struct DDS_DataWriter_get_listener
            (DDS_DataWriter this);

    DDS_Topic
        DDS_DataWriter_get_topic
            (DDS_DataWriter this);

    DDS_Publisher
        DDS_DataWriter_get_publisher
            (DDS_DataWriter this);

    DDS_ReturnCode_t
        DDS_DataWriter_wait_for_acknowledgments
            (DDS_DataWriter _this,
             const DDS_Duration_t *max_wait);

    DDS_ReturnCode_t
        DDS_DataWriter_get_liveliness_lost_status
            (DDS_DataWriter this,
             DDS_LivelinessLostStatus *status);

    DDS_ReturnCode_t
        DDS_DataWriter_get_offered_deadline_missed_status

```

```

        (DDS_DataWriter this,
         DDS_OfferedDeadlineMissedStatus *status);

DDS_ReturnCode_t
DDS_DataWriter_get_offered_incompatible_qos_status
(DDS_DataWriter this,
 DDS_OfferedIncompatibleQosStatus *status);

DDS_ReturnCode_t
DDS_DataWriter_get_publication_matched_status
(DDS_DataWriter this,
 DDS_PublicationMatchedStatus *status);

DDS_ReturnCode_t
DDS_DataWriter_assert_liveliness
(DDS_DataWriter this);

DDS_ReturnCode_t
DDS_DataWriter_get_matched_subscriptions
(DDS_DataWriter this,
 DDS_InstanceHandleSeq *subscription_handles);

DDS_ReturnCode_t
DDS_DataWriter_get_matched_subscription_data
(DDS_DataWriter this,
 DDS_SubscriptionBuiltinTopicData
 *subscription_data,
 const DDS_InstanceHandle_t subscription_handle
 );

```

The next paragraphs describe the usage of all `DDS_DataWriter` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited. The abstract operations are listed but not fully described because they are not implemented in this specific class. The full description of these operations is located in the subclasses, which contain the data type specific implementation of these operations.

3.4.2.2 DDS_DataWriter_assert_liveliness

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataWriter_assert_liveliness
(DDS_DataWriter _this);

```

Description

This operation asserts the liveliness for the `DDS_DataWriter`.

Parameters

in DDS_DataWriter_this - is the DDS_DataWriter object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_NOT_ENABLED.

Detailed Description

This operation will manually assert the liveness for the DDS_DataWriter. This way, the Data Distribution Service is informed that the corresponding DDS_DataWriter is still alive. This operation is used in combination with the DDS_LivelinessQosPolicy set to DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS or DDS_MANUAL_BY_TOPIC_LIVELINESS_QOS. See Section 3.1.3.10, *DDS_LivelinessQosPolicy*, on page 80, for more information on LivelinessQosPolicy.

Writing data via the DDS_DataWriter_write operation of a DDS_DataWriter will assert the liveness on the DDS_DataWriter itself and its containing DDS_DomainParticipant. Therefore, DDS_DataWriter_assert_liveliness is only needed when data is **not** written regularly.

The liveness should be asserted by the application, depending on the DDS_LivelinessQosPolicy. Asserting the liveness for this DDS_DataWriter can also be achieved by asserting the liveness to the DDS_DomainParticipant.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the liveness of this DDS_DataWriter has successfully been asserted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_DataWriter has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the DDS_DataWriter is not enabled.

3.4.2.3 DDS_DataWriter_dispose (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_dispose
        (DDS_DataWriter _this,
         const <data> *instance_data,
         const DDS_InstanceHandle_t instance_handle);
```

3.4.2.4 DDS_DataWriter_dispose_w_timestamp (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_dispose_w_timestamp
        (DDS_DataWriter _this,
         const <data> *instance_data,
         const DDS_InstanceHandle_t instance_handle,
         const DDS_Time_t *source_timestamp);
```

3.4.2.5 DDS_DataWriter_enable (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_enable
        (DDS_DataWriter _this);
```


3.4.2.6 DDS_DataWriter_get_key_value (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_key_value
        (DDS_DataWriter _this,
         <data> *key_holder,
         const DDS_InstanceHandle_t handle);
```

3.4.2.7 DDS_DataWriter_get_listener

Synopsis

```
#include <dds_dcps.h>
struct DDS_DataWriterListener
    DDS_DataWriter_get_listener
        (DDS_DataWriter _this);
```

Description

This operation allows access to a `DDS_DataWriterListener`.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

Return Value

struct `DDS_DataWriterListener` - a pointer to the `DDS_DataWriterListener` attached to the `DDS_DataWriter`.

Detailed Description

This operation allows access to a `DDS_DataWriterListener` attached to the `DDS_DataWriter`. When no `DDS_DataWriterListener` was attached to the `DDS_DataWriter`, the `DDS_OBJECT_NIL` pointer is returned.

3.4.2.8 DDS_DataWriter_get_liveliness_lost_status

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
```

```
DDS_DataWriter_get_liveliness_lost_status
(DDS_DataWriter _this,
 DDS_LivelinessLostStatus *status);
```

Description

This operation obtains the `DDS_LivelinessLostStatus` struct of the `DDS_DataWriter`.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

inout `DDS_LivelinessLostStatus *status` - the contents of the `DDS_LivelinessLostStatus` struct of the `DDS_DataWriter` will be copied into the location specified by `status`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are: `DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation obtains the `DDS_LivelinessLostStatus` struct of the `DDS_DataWriter`. This struct contains the information whether the liveliness (that the `DDS_DataWriter` has committed through its `DDS_LivelinessQosPolicy`) was respected.

This means that the status represents whether the `DDS_DataWriter` failed to actively signal its liveliness within the offered liveliness period. If the liveliness is lost, the `DDS_DataReader` objects will consider the `DDS_DataWriter` as no longer “alive”.

The `DDS_LivelinessLostStatus` can also be monitored using a `DDS_DataWriterListener` or by using the associated `DDS_StatusCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the current `DDS_LivelinessLostStatus` of this `DDS_DataWriter` has successfully been copied into the specified `status` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataWriter* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.2.9 DDS_DataWriter_get_matched_subscription_data

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_matched_subscription_data
        (DDS_DataWriter _this,
         DDS_SubscriptionBuiltinTopicData *subscription_data,
         const DDS_InstanceHandle_t subscription_handle);
```

Description

This operation retrieves information on the specified subscription that is currently “associated” with the *DDS_DataWriter*.

Parameters

in DDS_DataWriter _this - the *DDS_DataWriter* object on which the operation is operated.

*inout DDS_SubscriptionBuiltinTopicData *subscription_data* - a pointer to the sample in which the information about the specified subscription is to be stored.

in const DDS_InstanceHandle_t subscription_handle - a handle to the subscription whose information needs to be retrieved.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_NOT_ENABLED*.

Detailed Description

This operation retrieves information on the specified subscription that is currently “associated” with the *DDS_DataWriter*. That is, a subscription with a matching Topic and compatible QoS that the application has not indicated should be “ignored” by means of the *DDS_DomainParticipant_ignore_subscription* operation.

The `subscription_handle` must correspond to a subscription currently associated with the `DDS_DataWriter`, otherwise the operation will fail and return `DDS_RETCODE_BAD_PARAMETER`. The operation `DDS_DataWriter_get_matched_subscriptions` can be used to find the subscriptions that are currently matched with the `DDS_DataWriter`.

The operation may also fail if the infrastructure does not hold the information necessary to fill in the `subscription_data`. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the information on the specified subscription has been successfully retrieved.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_UNSUPPORTED` - OpenSplice is configured not to maintain the information about “associated” subscriptions.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_DataWriter` is not enabled.

3.4.2.10 `DDS_DataWriter_get_matched_subscriptions`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_matched_subscriptions
        (DDS_DataWriter _this,
         DDS_InstanceHandleSeq *subscription_handles);
```

Description

This operation retrieves the list of subscriptions currently "associated" with the `DDS_DataWriter`.

Parameters

in DDS_DataWriter _this - the DDS_DataWriter object on which the operation is operated.

*inout DDS_InstanceHandleSeq *subscription_handles* - a sequence which is used to pass the list of all associated subscriptions.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_NOT_ENABLED.

Detailed Description

This operation retrieves the list of subscriptions currently "associated" with the DDS_DataWriter. That is, subscriptions that have a matching Topic and compatible QoS that the application has not indicated should be "ignored" by means of the DDS_DomainParticipant_ignore_subscription operation.

The subscription_handles sequence and its buffer may be pre-allocated by the application and therefore must either be re-used in a subsequent invocation of the DDS_DataWriter_get_matched_subscriptions operation or be released by calling DDS_free on the returned subscription_handles. If the pre-allocated sequence is not big enough to hold the number of associated subscriptions, the sequence will automatically be (re-)allocated to fit the required size.

The handles returned in the subscription_handles sequence are the ones that are used by the DDS implementation to locally identify the corresponding matched DataReader entities. You can access more detailed information about a particular subscription by passing its subscription_handle to either the DDS_DataWriter_get_matched_subscription_data operation or to the DDS_SubscriptionBuiltinTopicDataReader_read_instance operation on the built-in reader for the "DCPSSubscription" topic.



Be aware that since DDS_InstanceHandle_t is an opaque datatype, it does not necessarily mean that the handles obtained from the DDS_DataWriter_get_matched_subscriptions operation have the same value as the ones that appear in the instance_handle field of the DDS_SampleInfo when retrieving the subscription info through corresponding "DCPSSubscriptions" built-in reader. You can't just compare two handles to determine whether they represent the same subscription. If you want to know whether two handles actually do represent the same subscription, use both handles to retrieve their corresponding DDS_SubscriptionBuiltinTopicData samples and then compare the key field of both samples.

The operation may fail if the infrastructure does not locally maintain the connectivity information. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the list of associated subscriptions has successfully been obtained.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_UNSUPPORTED` - OpenSplice is configured not to maintain the information about “associated” subscriptions.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_DataWriter` is not enabled.

3.4.2.11 `DDS_DataWriter_get_offered_deadline_missed_status`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_offered_deadline_missed_status
        (DDS_DataWriter _this,
         DDS_OfferedDeadlineMissedStatus *status);
```

Description

This operation obtains the `DDS_OfferedDeadlineMissedStatus` struct of the `DDS_DataWriter`.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

*inout DDS_OfferedDeadlineMissedStatus *status* - the contents of the DDS_OfferedDeadlineMissedStatus struct of the DDS_DataWriter will be copied into the location specified by *status*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation obtains the DDS_OfferedDeadlineMissedStatus struct of the DDS_DataWriter. This struct contains the information whether the deadline (that the DDS_DataWriter has committed through its DDS_DeadlineQosPolicy) was respected for each instance.

The DDS_OfferedDeadlineMissedStatus can also be monitored using a DDS_DataWriterListener or by using the associated DDS_StatusCondition.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the current DDS_LivelinessLostStatus of this DDS_DataWriter has successfully been copied into the specified status parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_DataWriter has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.2.12 DDS_DataWriter_get_offered_incompatible_qos_status

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_offered_incompatible_qos_status
        (DDS_DataWriter _this,
         DDS_OfferedIncompatibleQosStatus *status);
```

Description

This operation obtains the `DDS_OfferedIncompatibleQosStatus` struct of the `DDS_DataWriter`.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

inout `DDS_OfferedIncompatibleQosStatus *status` - the contents of the `DDS_OfferedIncompatibleQosStatus` struct of the `DDS_DataWriter` will be copied into the location specified by `status`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are: `DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation obtains the `DDS_OfferedIncompatibleQosStatus` struct of the `DDS_DataWriter`. This struct contains the information whether a `QosPolicy` setting was incompatible with the requested `QosPolicy` setting.

This means that the status represents whether a `DDS_DataReader` object has been discovered by the `DDS_DataWriter` with the same `DDS_Topic` and a requested `DDS_DataReaderQos` that was incompatible with the one offered by the `DDS_DataWriter`.

The `DDS_OfferedIncompatibleQosStatus` can also be monitored using a `DDS_DataWriterListener` or by using the associated `DDS_StatusCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the current `DDS_OfferedIncompatibleQosStatus` of this `DDS_DataWriter` has successfully been copied into the specified `status` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataWriter` has already been deleted.

- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.2.13 DDS_DataWriter_get_publication_matched_status

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_publication_matched_status
        (DDS_DataWriter _this,
         DDS_PublicationMatchedStatus *status);
```

Description

This operation obtains the *DDS_PublicationMatchedStatus* struct of the *DDS_DataWriter*.

Parameters

in DDS_DataWriter _this - the *DDS_DataWriter* object on which the operation is operated.

*inout DDS_PublicationMatchedStatus *status* - the contents of the *DDS_PublicationMatchedStatus* struct of the *DDS_DataWriter* will be copied into the location specified by *status*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation obtains the *DDS_PublicationMatchedStatus* struct of the *DDS_DataWriter*. This struct contains information about whether a new match has been discovered for the current publication, or whether an existing match has ceased to exist.

This means that the status represents that either a *DataReader* object has been discovered by the *DDS_DataWriter* with the same Topic and a compatible Qos, or that a previously-discovered *DataReader* has ceased to be matched to the current *DDS_DataWriter*. A *DataReader* may cease to match when it gets deleted, when it changes its Qos to a value that is incompatible with the current *DDS_DataWriter* or when either the *DDS_DataWriter* or the *DataReader* has chosen to put its

matching counterpart on its ignore-list using the `DDS_DomainParticipant_ignore_subscription` or `DDS_DomainParticipant_ignore_publication` operations.

The operation may fail if the infrastructure does not hold the information necessary to fill in the `DDS_PublicationMatchedStatus`. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

The `DDS_PublicationMatchedStatus` can also be monitored using a `DDS_DataWriterListener` or by using the associated `DDS_StatusCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the current `DDS_PublicationMatchedStatus` of this `DDS_DataWriter` has successfully been copied into the specified status parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_UNSUPPORTED` - OpenSplice is configured not to maintain the information about “associated” subscriptions.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.4.2.14 `DDS_DataWriter_get_publisher`

Synopsis

```
#include <dds_dcps.h>
DDS_Publisher
    DDS_DataWriter_get_publisher
        (DDS_DataWriter _this);
```

Description

This operation returns the `DDS_Publisher` to which the `DDS_DataWriter` belongs.

Parameters

in DDS_DataWriter _this - the DDS_DataWriter object on which the operation is operated.

Return Value

DDS_Publisher - a pointer to the DDS_Publisher to which the DDS_DataWriter belongs.

Detailed Description

This operation returns the DDS_Publisher to which the DDS_DataWriter belongs, thus the DDS_Publisher that has created the DDS_DataWriter. If the DDS_DataWriter is already deleted, the DDS_OBJECT_NIL pointer is returned.

3.4.2.15 DDS_DataWriter_get_qos**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_get_qos
        (DDS_DataWriter _this,
         DDS_DataWriterQos *qos);
```

Description

This operation allows access to the existing list of QosPolicy settings for a DDS_DataWriter.

Parameters

in DDS_DataWriter _this - the DDS_DataWriter object on which the operation is operated.

*inout DDS_DataWriterQos *qos* - a pointer to the destination DDS_DataWriterQos struct in which the QosPolicy settings will be copied.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation allows access to the existing list of QosPolicy settings of a DDS_DataWriter on which this operation is used. This DDS_DataWriterQos is stored at the location pointed to by the qos parameter.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the existing set of QoS policy values applied to this *DDS_DataWriter* has successfully been copied into the specified *DDS_DataWriterQos* parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataWriter* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.4.2.16 DDS_DataWriter_get_status_changes (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
    DDS_DataWriter_get_status_changes
        (DDS_DataWriter _this);
```

3.4.2.17 DDS_DataWriter_get_statuscondition (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
    DDS_DataWriter_get_statuscondition
        (DDS_DataWriter _this);
```

3.4.2.18 DDS_DataWriter_get_topic**Synopsis**

```
#include <dds_dcps.h>
DDS_Topic
    DDS_DataWriter_get_topic
        (DDS_DataWriter _this);
```

Description

This operation returns the `DDS_Topic` which is associated with the `DDS_DataWriter`.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

Return Value

`DDS_Topic` - Return value is a pointer to the `DDS_Topic` which is associated with the `DDS_DataWriter`.

Detailed Description

This operation returns the `DDS_Topic` which is associated with the `DDS_DataWriter`, thus the `DDS_Topic` with which the `DDS_DataWriter` is created. If the `DDS_DataWriter` is already deleted, the `DDS_OBJECT_NIL` pointer is returned.

3.4.2.19 DDS_DataWriter_lookup_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_InstanceHandle_t
    DDS_DataWriter_lookup_instance
        (DDS_DataWriter _this,
         <data> *instance_data);
```

3.4.2.20 DDS_DataWriter_register_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
const DDS_InstanceHandle_t
```

```
DDS_DataWriter_register_instance
(DDS_DataWriter _this,
 const <data> *instance_data);
```

3.4.2.21 DDS_DataWriter_register_instance_w_timestamp (abstract)

This abstract operation is defined as a generic operation, which is implemented by the <NameSpace>_<type>DataWriter class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type Foo (defined in the module SPACE) derived SPACE_FooDataWriter class.

Synopsis

```
#include <dds_dcps.h>
const DDS_InstanceHandle_t
    DDS_DataWriter_register_instance_w_timestamp
    (DDS_DataWriter _this,
     const <data> *instance_data,
     const DDS_Time_t *source_timestamp);
```

3.4.2.22 DDS_DataWriter_set_listener

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_set_listener
    (DDS_DataWriter _this,
     const struct DDS_DataWriterListener *a_listener,
     const DDS_StatusMask mask);
```

Description

This operation attaches a DDS_DataWriterListener to the DDS_DataWriter.

Parameters

in DDS_DataWriter _this - the DDS_DataWriter object on which the operation is operated.

*in const struct DDS_DataWriterListener *a_listener* - a pointer to the DDS_DataWriterListener instance, which will be attached to the DDS_DataWriter.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the DDS_DataWriterListener for a certain status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation attaches a *DDS_DataWriterListener* to the *DDS_DataWriter*. Only one *DDS_DataWriterListener* can be attached to each *DDS_DataWriter*. If a *DDS_DataWriterListener* was already attached, the operation will replace it with the new one. When *a_listener* is the *DDS_OBJECT_NIL* pointer, it represents a listener that is treated as a NOOP¹ for all statuses activated in the bitmask.

Communication Status

For each communication status, the *StatusChangedFlag* flag is initially set to *FALSE*. It becomes *TRUE* whenever that communication status changes. For each communication status activated in the mask, the associated *DDS_DataWriterListener* operation is invoked and the communication status is reset to *FALSE*, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the *get_<status_name>_status* from inside the listener it will see the status already reset. An exception to this rule is the *DDS_OBJECT_NIL* listener, which does not reset the communication statuses for which it is invoked.

The following statuses are applicable to the *DDS_DataWriterListener*:

- *DDS_OFFERED_DEADLINE_MISSED_STATUS*
- *DDS_OFFERED_INCOMPATIBLE_QOS_STATUS*
- *DDS_LIVELINESS_LOST_STATUS*
- *DDS_PUBLICATION_MATCHED_STATUS*.



Be aware that the *DDS_PUBLICATION_MATCHED_STATUS* is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when *OpenSplice* is configured not to maintain discovery information in the *Networking Service*. (See the description for the *NetworkingService/Discovery/enabled* property in the *Deployment Manual* for more information about this subject.) In this case the operation will return *DDS_RETCODE_UNSUPPORTED*.

1. Short for **No-Operation**, an instruction that performs nothing at all.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

In case a communication status is not activated in the mask of the `DDS_DataWriterListener`, the `DDS_PublisherListener` of the containing `DDS_Publisher` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_PublisherListener` of the containing `DDS_Publisher` and a `DDS_DataWriter` specific behaviour when needed. In case the communication status is not activated in the mask of the `DDS_PublisherListener` as well, the communication status will be propagated to the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant`. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_DataWriterListener` is attached.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_UNSUPPORTED` - a status was selected that cannot be supported because the infrastructure does not maintain the required connectivity information.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.4.2.23 `DDS_DataWriter_set_qos`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
```



```
DDS_DataWriter_set_qos
(DDS_DataWriter _this,
 const DDS_DataWriterQos *qos);
```

Description

This operation replaces the existing set of `QosPolicy` settings for a `DDS_DataWriter`.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

in `const DDS_DataWriterQos *qos` - contain the new set of `QosPolicy` settings for the `DDS_DataWriter`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_UNSUPPORTED`, `DDS_RETCODE_ALLREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_IMMUTABLE_POLICY` or `DDS_RETCODE_INCONSISTENT_POLICY`.

Detailed Description

This operation replaces the existing set of `QosPolicy` settings for a `DDS_DataWriter`. The parameter `qos` contains the struct with the `QosPolicy` settings which is checked for self-consistency and mutability. When the application tries to change a `QosPolicy` setting for an enabled `DDS_DataWriter`, which can only be set before the `DDS_DataWriter` is enabled, the operation will fail and a `DDS_RETCODE_IMMUTABLE_POLICY` is returned. In other words, the application must provide the presently set `QosPolicy` settings in case of the immutable `QosPolicy` settings. Only the mutable `QosPolicy` settings can be changed. When `qos` contains conflicting `QosPolicy` settings (not self-consistent), the operation will fail and a `DDS_RETCODE_INCONSISTENT_POLICY` is returned.

The set of `QosPolicy` settings specified by the `qos` parameter are applied on top of the existing `QoS`, replacing the values of any policies previously set (provided, the operation returned `DDS_RETCODE_OK`).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the new default `DDS_DataWriterQos` is set.
- `DDS_RETCODE_ERROR` - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *qos* is not a valid *DDS_DataWriterQos*. It contains a *QosPolicy* setting with an invalid *DDS_Duration_t* value, an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- *DDS_RETCODE_UNSUPPORTED* - one or more of the selected *QosPolicy* values are currently not supported by OpenSplice.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataWriter* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_IMMUTABLE_POLICY* - the parameter *qos* contains an immutable *QosPolicy* setting with a different value than set during enabling of the *DDS_DataWriter*.
- *DDS_RETCODE_INCONSISTENT_POLICY* - the parameter *qos* contains conflicting *QosPolicy* settings, e.g. a history depth that is higher than the specified resource limits.

3.4.2.24 DDS_DataWriter_unregister_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_unregister_instance
        (DDS_DataWriter _this,
         const <data> *instance_data,
         const DDS_InstanceHandle_t handle);
```

3.4.2.25 DDS_DataWriter_unregister_instance_w_timestamp (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_unregister_instance_w_timestamp
        (DDS_DataWriter _this,
         const <data> *instance_data,
         const DDS_InstanceHandle_t handle,
         const DDS_Time_t *source_timestamp);
```

3.4.2.26 DDS_DataWriter_wait_for_acknowledgments

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_wait_for_acknowledgments
        (DDS_DataWriter _this,
         const DDS_Duration_t *max_wait);
```

Description

This operation blocks the calling thread until either all data written by the `DDS_DataWriter` is acknowledged by the local infrastructure, or until the duration specified by `max_wait` parameter elapses, whichever happens first.

Parameters

in `DDS_DataWriter _this` - the `DDS_DataWriter` object on which the operation is operated.

in `const DDS_Duration_t *max_wait` - the maximum duration to block for the `DDS_DataWriter_wait_for_acknowledgments`, after which the application thread is unblocked. The special constant `DDS_DURATION_INFINITE` can be used when the maximum waiting time does not need to be bounded.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED` or `DDS_RETCODE_TIMEOUT`.

Detailed Description

This operation blocks the calling thread until either all data written by the `DDS_DataWriter` is acknowledged by the local infrastructure, or until the duration specified by `max_wait` parameter elapses, whichever happens first.

Data is acknowledged by the local infrastructure when it does not need to be stored in its `DataWriter`'s local history. When a locally-connected subscription (including the networking service) has no more resources to store incoming samples it will start to reject these samples, resulting in its source `DataWriter` to store them temporarily in its own local history to be retransmitted at a later moment in time. In such scenarios, the `DDS_DataWriter_wait_for_acknowledgments` operation will block until the `DDS_DataWriter` has retransmitted its entire history, which is therefore effectively empty, or until the `max_wait` timeout expires, whichever happens first. In the first case the operation will return `DDS_RETCODE_OK`, in the latter it will return `DDS_RETCODE_TIMEOUT`.



Be aware that in case the operation returns `DDS_RETCODE_OK`, the data has only been acknowledged by the local infrastructure: it does not mean all remote subscriptions have already received the data. However, delivering the data to remote nodes is then the sole responsibility of the networking service: even when the publishing application would terminate, all data that has not yet been received may be considered 'on-route' and will therefore eventually arrive (unless the networking service itself crashes). In contrast, if the `DDS_DataWriter` would still have data in its local history buffer when it terminates, this data is considered 'lost'.

This operation is intended to be used only if the `DDS_DataWriter` has its `DDS_ReliabilityQosPolicyKind` set to `DDS_RELIABLE_RELIABILITY_QOS`. Otherwise the operation will return immediately with `DDS_RETCODE_OK`, since best-effort `DataWriters` will never store rejected samples in their local history: they will just drop them and continue business as usual.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the data of the `DDS_DataWriter` has been acknowledged by the local infrastructure.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_DataWriter` is not enabled.
- `DDS_RETCODE_TIMEOUT` - not all data is acknowledged before `max_wait` elapsed.

3.4.2.27 DDS_DataWriter_write (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<Namespace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_write
    (DDS_DataWriter _this,
     const <data> *instance_data,
     const DDS_InstanceHandle_t handle);
```

3.4.2.28 DDS_DataWriter_write_w_timestamp (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<Namespace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_write_w_timestamp
    (DDS_DataWriter _this,
     const <data> *instance_data,
     const DDS_InstanceHandle_t handle,
     const DDS_Time_t *source_timestamp);
```

3.4.2.29 DDS_DataWriter_writedispose (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<Namespace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataWriter_writedispose
    (DDS_DataWriter _this,
     const <data> *instance_data,
```

```
const DDS_InstanceHandle_t instance_handle);
```

3.4.2.30 DDS_DataWriter_writedispose_w_timestamp (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataWriter` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataWriter` class.

Synopsis

```
#include <dds_dcps.h>
        DDS_ReturnCode_t
        DDS_DataWriter_writedispose_w_timestamp
        (DDS_DataWriter _this,
         const <data> *instance_data,
         const DDS_InstanceHandle_t instance_handle,
         const DDS_Time_t *source_timestamp);
```

3.4.2.31 Class SPACE_FooDataWriter

The pre-processor generates from IDL type descriptions the application `<NameSpace>_<type>DataWriter` classes. For each application data type that is used as `DDS_Topic` data type, a typed class `<NameSpace>_<type>DataWriter` is derived from the `DDS_DataWriter` class. In this paragraph, the class `SPACE_FooDataWriter` describes the operations of these derived `<NameSpace>_<type>DataWriter` classes as an example for the fictional application type `Foo` (defined in the module `SPACE`).

For instance, for an application, the definitions are located in the `Space.idl` file. The pre-processor will generate a `Space.h` include file.

A `SPACE_FooDataWriter` is attached to exactly one `DDS_Publisher` which acts as a factory for it. The `SPACE_FooDataWriter` is bound to exactly one `DDS_Topic` that has been registered to use a data type `Foo` (defined in the module `SPACE`). The `DDS_Topic` must exist prior to the `SPACE_FooDataWriter` creation.

The interface description of this class is as follows:

```
/*
 * interface SPACE_FooDataWriter
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *     SPACE_FooDataWriter_get_statuscondition
 *         (SPACE_FooDataWriter _this);
 */
/* DDS_StatusMask
```

```

    *     SPACE_FooDataWriter_get_status_changes
    *     (SPACE_FooDataWriter _this);
    */
/* DDS_ReturnCode_t
    *     SPACE_FooDataWriter_enable
    *     (SPACE_FooDataWriter _this);
    */
/*
    * inherited from class DDS_DataWriter
    */
/* DDS_ReturnCode_t
    *     SPACE_FooDataWriter_set_qos
    *     (SPACE_FooDataWriter _this,
    *      const DDS_DataWriterQos *qos);
    */

/* DDS_ReturnCode_t
    *     SPACE_FooDataWriter_get_qos
    *     (SPACE_FooDataWriter _this,
    *      SPACE_FooDataWriterQos *qos);
    */

/* DDS_ReturnCode_t
    *     SPACE_FooDataWriter_set_listener
    *     (SPACE_FooDataWriter _this,
    *      const struct DDS_DataWriterListener *a_listener,
    *      const DDS_StatusMask mask);
    */

/* struct SPACE_FooDataWriterListener
    *     SPACE_FooDataWriter_get_listener
    *     (SPACE_FooDataWriter _this);
    */

/* DDS_Topic
    *     SPACE_FooDataWriter_get_topic
    *     (SPACE_FooDataWriter _this);
    */

/* DDS_Publisher
    *     SPACE_FooDataWriter_get_publisher
    *     (SPACE_FooDataWriter _this);
    */

/* DDS_ReturnCode_t
    *     SPACE_FooDataWriter_wait_for_acknowledgments
    *     (DDS_DataWriter _this,
    *      const DDS_Duration_t *max_wait);
    */

```

```

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_get_liveliness_lost_status
 *   (SPACE_FooDataWriter _this,
 *    DDS_LivelinessLostStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_get_offered_deadline_missed_status
 *   (SPACE_FooDataWriter _this,
 *    DDS_OfferedDeadlineMissedStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_get_offered_incompatible_qos_status
 *   (SPACE_FooDataWriter _this,
 *    DDS_OfferedIncompatibleQosStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_get_publication_matched_status
 *   (SPACE_FooDataWriter _this,
 *    DDS_PublicationMatchedStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_assert_liveliness
 *   (SPACE_FooDataWriter _this);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_get_matched_subscriptions
 *   (SPACE_FooDataWriter _this,
 *    DDS_InstanceHandleSeq *subscription_handles);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataWriter_get_matched_subscription_data
 *   (SPACE_FooDataWriter _this,
 *    DDS_SubscriptionBuiltinTopicData *subscription_data,
 *    const DDS_InstanceHandle_t subscription_handle);
 */
/*
 * implemented API operations
 */
DDS_InstanceHandle_t
SPACE_FooDataWriter_register_instance
(SPACE_FooDataWriter _this,
 const Foo *instance_data);
DDS_InstanceHandle_t
SPACE_FooDataWriter_register_instance_w_timestamp

```



```

        (SPACE_FooDataWriter _this,
         const Foo *instance_data,
         const DDS_Time_t *source_timestamp);
DDS_ReturnCode_t
SPACE_FooDataWriter_unregister_instance
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
SPACE_FooDataWriter_unregister_instance_w_timestamp
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t handle,
     const DDS_Time_t *source_timestamp);
DDS_ReturnCode_t
SPACE_FooDataWriter_write
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t handle);
DDS_ReturnCode_t
SPACE_FooDataWriter_write_w_timestamp
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t handle,
     const DDS_Time_t *source_timestamp);
DDS_ReturnCode_t
SPACE_FooDataWriter_dispose
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t instance_handle);
DDS_ReturnCode_t
SPACE_FooDataWriter_dispose_w_timestamp
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t instance_handle,
     const DDS_Time_t *source_timestamp);
DDS_ReturnCode_t
SPACE_FooDataWriter_writedispose
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t instance_handle);
DDS_ReturnCode_t
SPACE_FooDataWriter_writedispose_w_timestamp
    (SPACE_FooDataWriter _this,
     const Foo *instance_data,
     const DDS_InstanceHandle_t instance_handle,
     const DDS_Time_t *source_timestamp);
DDS_ReturnCode_t
SPACE_FooDataWriter_get_key_value
    (SPACE_FooDataWriter _this,

```

```

        Foo *key_holder,
        const DDS_InstanceHandle_t handle);
    DDS_InstanceHandle_t
    SPACE_FooDataWriter_lookup_instance
    (SPACE_FooDataWriter _this,
     Foo *instance_data);

```

The next paragraphs describe the usage of all `SPACE_FooDataWriter` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.4.2.32 `SPACE_FooDataWriter_assert_liveliness` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataWriter` for further explanation.

Synopsis

```

#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_assert_liveliness
(SPACE_FooDataWriter _this);

```

3.4.2.33 `SPACE_FooDataWriter_dispose`

Synopsis

```

#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_dispose
(SPACE_FooDataWriter _this,
 const Foo *instance_data,
 const DDS_InstanceHandle_t instance_handle);

```

Description

This operation requests the Data Distribution Service to mark the instance for deletion.

Parameters

in `SPACE_FooDataWriter _this` - the `SPACE_FooDataWriter` object on which the operation is operated.

in `const Foo *instance_data` - the actual instance to be disposed of.

in `const DDS_InstanceHandle_t instance_handle` - the handle to the instance to be disposed of.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_TIMEOUT.

Detailed Description

This operation requests the Data Distribution Service to mark the instance for deletion. Copies of the instance and its corresponding samples, which are stored in every connected *DDS_DataReader* and, dependent on the *QoSPolicy* settings, also in the Transient and Persistent stores, will be marked for deletion by setting their *DDS_InstanceStateKind* to *DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE*.

When this operation is used, the Data Distribution Service will automatically supply the value of the *source_timestamp* that is made available to connected *DDS_DataReader* objects. This timestamp is important for the interpretation of the *DDS_DestinationOrderQoSPolicy*.

As a side effect, this operation asserts liveness on the *DDS_DataWriter* itself and on the containing *DDS_DomainParticipant*.

Effects on DataReaders

Actual deletion of the instance administration in a connected *DDS_DataReader* will be postponed until the following conditions have been met:

- the instance must be unregistered (either implicitly or explicitly) by all connected *DDS_DataWriters* that have previously registered it
 - A *DDS_DataWriter* can register an instance explicitly by using one of the special operations *SPACE_FooDataWriter_register_instance* or *SPACE_FooDataWriter_register_instance_w_timestamp*.
 - A *DDS_DataWriter* can register an instance implicitly by using the special constant *DDS_HANDLE_NIL* in any of the other *DDS_DataWriter* operations.
 - A *DDS_DataWriter* can unregister an instance explicitly by using one of the special operations *SPACE_FooDataWriter_unregister_instance* or *SPACE_FooDataWriter_unregister_instance_w_timestamp*.
 - A *DDS_DataWriter* will unregister all its contained instances implicitly when it is deleted.

- When a `DDS_DataReader` detects a loss of liveliness in one of its connected `DDS_DataWriters`, it will consider all instances registered by that `DDS_DataWriter` as being implicitly unregistered.
- **and** the application must have consumed all samples belonging to the instance, either implicitly or explicitly.
 - An application can consume samples explicitly by invoking the `SPACE_FooDataReader_take` operation, or one of its variants.
 - The `DDS_DataReader` can consume disposed samples implicitly when the `autopurge_disposed_samples_delay` of the `DDS_ReaderDataLifecycleQosPolicy` has expired.

The `DDS_DataReader` may also remove instances that haven't been disposed first: this happens when the `autopurge_nowriter_samples_delay` of the `DDS_ReaderDataLifecycleQosPolicy` has expired after the instance is considered unregistered by all connected `DDS_DataWriters` (i.e. when it has a `DDS_InstanceStateKind` of `DDS_NOT_ALIVE_NO_WRITERS`). See also Section 3.1.3.15, *DDS_ReaderDataLifecycleQosPolicy*, on page 92.

Effects on Transient/Persistent Stores

Actual deletion of the instance administration in the connected Transient and Persistent stores will be postponed until the following conditions have been met:

- the instance must be unregistered (either implicitly or explicitly) by all connected `DDS_DataWriters` that have previously registered it. (See above.)
- **and** the period of time specified by the `service_cleanup_delay` attribute in the `DDS_DurabilityServiceQosPolicy` on the `DDS_Topic` must have elapsed after the instance is considered unregistered by all connected `DDS_DataWriters`.

See also Section 3.1.3.4, *DDS_DurabilityServiceQosPolicy*, on page 71.

Instance Handle

The `DDS_HANDLE_NIL` handle value can be used for the parameter `instance_handle`. This indicates the identity of the instance is automatically deduced from the `instance_data` (by means of the key).

If `instance_handle` is any value other than `DDS_HANDLE_NIL`, then it must correspond to the value that was returned by either the `SPACE_FooDataWriter_register_instance` operation or the `SPACE_FooDataWriter_register_instance_w_timestamp` operation when the instance (identified by its key) was registered. If there is no correspondence, then the result of the operation is unspecified.

The sample that is passed as `instance_data` is only used to check for consistency between its key values and the supplied `instance_handle`: the sample itself will not actually be delivered to the connected `DDS_DataReaders`. Use the `SPACE_FooDataWriter_writedispose` operation if the sample itself should be delivered together with the dispose request.

Blocking

If the `DDS_HistoryQosPolicy` is set to `DDS_KEEP_ALL_HISTORY_QOS`, the `SPACE_FooDataWriter_dispose` operation on the `DDS_DataWriter` may block if the modification would cause data to be lost because one of the limits, specified in the `DDS_ResourceLimitsQosPolicy`, to be exceeded. Under these circumstances, the `max_blocking_time` attribute of the `ReliabilityQosPolicy` configures the maximum time the `SPACE_FooDataWriter_dispose` operation may block (waiting for space to become available). If `max_blocking_time` elapses before the `DDS_DataWriter` is able to store the modification without exceeding the limits, the `SPACE_FooDataWriter_dispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Sample Validation

OpenSplice DDS offers the possibility to check the sample that is passed as `instance_data` for validity. Because validity checking might reduce the overall performance, it is by default disabled. This has been done by enclosing the validity checking with conditional compiler directives like this:

```
#ifdef OSPL_BOUNDS_CHECK
    // check a specific bound.
#endif
```

i

By defining a macro called `OSPL_OSPL_BOUNDS_CHECK`, the validity checking will be included. On most compilers this macro can be defined by passing an additional command line parameter called `-DOSPL_BOUNDS_CHECK`.

Since the `SPACE_FooDataWriter_dispose` operation only uses the sample to check for consistency between its key values and the supplied `instance_handle`, only those keyfields will be validated against the restrictions imposed by the IDL to C language mapping, where:

- an enum may not exceed the value of its highest label
- a string (bounded or unbounded) may not be NULL. (Use "" for an empty string instead)
- the length of a bounded string may not exceed the limit specified in IDL

If any of these restrictions is violated when validity checking is enabled, then the operation will fail and return a `DDS_RETCODE_BAD_PARAMETER`. More specific information about the context of this error will be written to the error log. When validity checking is disabled, any of these violations may result in undefined behaviour.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the Data Distribution Service is informed that the instance data must be disposed of.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - `instance_handle` is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the `instance_handle` has not been registered with this `SPACE_FooDataWriter`.
- `DDS_RETCODE_TIMEOUT` - the current action overflowed the available resources as specified by the combination of the `DDS_ReliabilityQosPolicy`, `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy`. This caused blocking of the `SPACE_FooDataWriter_dispose` operation, which could not be resolved before `max_blocking_time` of the `DDS_ReliabilityQosPolicy` elapsed.

3.4.2.34 `SPACE_FooDataWriter_dispose_w_timestamp`

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_dispose_w_timestamp
(SPACE_FooDataWriter _this,
 const Foo *instance_data,
 const DDS_InstanceHandle_t instance_handle,
 const DDS_Time_t *source_timestamp);
```

Description

This operation requests the Data Distribution Service to mark the instance for deletion and provides a value for the `source_timestamp` explicitly.

Parameters

in `SPACE_FooDataWriter _this` - the `SPACE_FooDataWriter` object on which the operation is operated.

in `const Foo *instance_data` - the actual instance to be disposed of.

in `const DDS_InstanceHandle_t instance_handle` - the handle to the instance to be disposed of.

in `const DDS_Time_t *source_timestamp` - the timestamp which is provided for the `DDS_DataReader`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED`, `DDS_RETCODE_PRECONDITION_NOT_MET` or `DDS_RETCODE_TIMEOUT`.

Detailed Description

This operation performs the same functions as `SPACE_FooDataWriter_dispose` except that the application provides the value for the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the Data Distribution Service is informed that the instance data must be disposed of.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - `instance_handle` is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.

- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataWriter* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the *instance_handle* has not been registered with this *SPACE_FooDataWriter*.
- *DDS_RETCODE_TIMEOUT* - the current action overflowed the available resources as specified by the combination of the *DDS_ReliabilityQosPolicy*, *DDS_HistoryQosPolicy* and *DDS_ResourceLimitsQosPolicy*. This caused blocking of the *SPACE_FooDataWriter_dispose_w_timestamp* operation, which could not be resolved before *max_blocking_time* of the *DDS_ReliabilityQosPolicy* elapsed.

3.4.2.35 *SPACE_FooDataWriter_enable* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_enable
        (SPACE_FooDataWriter _this);
```

3.4.2.36 *SPACE_FooDataWriter_get_key_value*

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_get_key_value
        (SPACE_FooDataWriter _this,
         Foo *key_holder,
         const DDS_InstanceHandle_t handle);
```

Description

This operation retrieves the key value of a specific instance.

Parameters

in SPACE_FooDataWriter _this - the *SPACE_FooDataWriter* object on which the operation is operated.

*inout Foo *key_holder* - the sample in which the key values are stored.

in const DDS_InstanceHandle_t handle - the handle to the instance from which to get the key value.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation retrieves the key value of the instance pointed to by *instance_handle*. When the operation is called with an *DDS_HANDLE_NIL* handle value as an *instance_handle*, the operation will return *DDS_RETCODE_BAD_PARAMETER*. The operation will only fill the fields that form the key inside the *key_holder* instance. This means that the non-key fields are not applicable and may contain garbage.

The operation must only be called on registered instances. Otherwise the operation returns the error *DDS_RETCODE_PRECONDITION_NOT_MET*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the *key_holder* instance contains the key values of the instance.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - *handle* is not a valid handle or *key_holder* is not a valid pointer.
- *DDS_RETCODE_ALREADY_DELETED* - the *SPACE_FooDataWriter* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataWriter* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - this instance is not registered.

3.4.2.37 *SPACE_FooDataWriter_get_listener* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_DataWriter* for further explanation.

Synopsis

```
#include <Space.h>
```

```
struct SPACE_FooDataWriterListener
    SPACE_FooDataWriter_get_listener
    (SPACE_FooDataWriter _this);
```

3.4.2.38 SPACE_FooDataWriter_get_liveliness_lost_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_get_liveliness_lost_status
    (SPACE_FooDataWriter _this,
     DDS_LivelinessLostStatus *status);
```

3.4.2.39 SPACE_FooDataWriter_get_matched_subscription_data (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_get_matched_subscription_data
    (SPACE_FooDataWriter _this,
     DDS_SubscriptionBuiltinTopicData *subscription_data,
     const DDS_InstanceHandle_t subscription_handle);
```

3.4.2.40 SPACE_FooDataWriter_get_matched_subscriptions (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_get_matched_subscriptions
    (SPACE_FooDataWriter _this,
     DDS_InstanceHandleSeq *subscription_handles);
```

3.4.2.41 SPACE_FooDataWriter_get_offered_deadline_missed_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```
#include <Space.h>
```

```

DDS_ReturnCode_t
SPACE_FooDataWriter_get_offered_deadline_missed_status
(SPACE_FooDataWriter _this,
 DDS_OfferedDeadlineMissedStatus *status);

```

3.4.2.42 **SPACE_FooDataWriter_get_offered_incompatible_qos_status (inherited)**

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```

#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_get_offered_incompatible_qos_status
(SPACE_FooDataWriter _this,
 DDS_OfferedIncompatibleQosStatus *status);

```

3.4.2.43 **SPACE_FooDataWriter_get_publication_matched_status (inherited)**

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```

#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_get_publication_matched_status
(SPACE_FooDataWriter _this,
 DDS_PublicationMatchedStatus *status);

```

3.4.2.44 **SPACE_FooDataWriter_get_publisher (inherited)**

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```

#include <Space.h>
DDS_Publisher
SPACE_FooDataWriter_get_publisher
(SPACE_FooDataWriter _this);

```

3.4.2.45 **SPACE_FooDataWriter_get_qos (inherited)**

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```

#include <Space.h>
DDS_ReturnCode_t

```

```
SPACE_FooDataWriter_get_qos
(SPACE_FooDataWriter _this,
 SPACE_FooDataWriterQos *qos);
```

3.4.2.46 SPACE_FooDataWriter_get_status_changes (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```
#include <Space.h>
DDS_StatusMask
SPACE_FooDataWriter_get_status_changes
(SPACE_FooDataWriter _this);
```

3.4.2.47 SPACE_FooDataWriter_get_statuscondition (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```
#include <Space.h>
DDS_StatusCondition
SPACE_FooDataWriter_get_statuscondition
(SPACE_FooDataWriter _this);
```

3.4.2.48 SPACE_FooDataWriter_get_topic (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```
#include <Space.h>
DDS_Topic
SPACE_FooDataWriter_get_topic
(SPACE_FooDataWriter _this);
```

3.4.2.49 SPACE_FooDataWriter_lookup_instance

Synopsis

```
#include <Space.h>
DDS_InstanceHandle_t
SPACE_FooDataWriter_lookup_instance
(SPACE_FooDataWriter _this,
 Foo *instance_data);
```

Description

This operation returns the value of the instance handle which corresponds to the `instance_data`.

Parameters

in `SPACE_FooDataWriter _this` - the `SPACE_FooDataWriter` object on which the operation is operated.

in `Foo *instance_data` - the instance for which the corresponding instance handle needs to be looked up.

Return Value

`DDS_InstanceHandle_t` - Result value is the instance handle which corresponds to the `instance_data`.

Detailed Description

This operation returns the value of the instance handle which corresponds to the `instance_data`. The `instance_data` parameter is only used for the purpose of examining the fields that define the key. The instance handle can be used in any write, dispose or unregister operations (or their timestamped variants) that operate on a specific instance. Note that `DDS_DataWriter` instance handles are local, and are not interchangeable with `DDS_DataReader` instance handles nor with instance handles of an other `DDS_DataWriter`.

This operation does not register the instance in question. If the instance has not been previously registered, if the `DDS_DataWriter` is already deleted or if for any other reason the Service is unable to provide an instance handle, the Service will return the special value `DDS_HANDLE_NIL`.

Sample Validation

OpenSplice DDS offers the possibility to check the sample that is passed as `instance_data` for validity. Because validity checking might reduce the overall performance, it is by default disabled. This has been done by enclosing the validity checking with conditional compiler directives like this:

```
#ifdef OSPL_BOUNDS_CHECK
    // check a specific bound.
#endif
```

i

By defining a macro called `OSPL_OSPL_BOUNDS_CHECK`, the validity checking will be included. On most compilers this macro can be defined by passing an additional command line parameter called `-DOSPL_BOUNDS_CHECK`.

Since the `SPACE_FooDataWriter_lookup_instance` operation merely uses the sample to determine its identity based on the uniqueness of its key values, only the keyfields will be validated against the restrictions imposed by the IDL to C language mapping, where:

- an enum may not exceed the value of its highest label
- a string (bounded or unbounded) may not be `NULL`. (Use `""` for an empty string instead)
- the length of a bounded string may not exceed the limit specified in IDL

If any of these restrictions is violated when validity checking is enabled, then the operation will fail and return a `DDS_HANDLE_NIL`. More specific information about the context of this error will be written to the error log. When validity checking is disabled, any of these violations may result in undefined behaviour.

3.4.2.50 `SPACE_FooDataWriter_register_instance`

Synopsis

```
#include <Space.h>
        DDS_InstanceHandle_t
        SPACE_FooDataWriter_register_instance
        (SPACE_FooDataWriter _this,
         const Foo *instance_data);
```

Description

This operation informs the Data Distribution Service that the application will be modifying a particular instance.

Parameters

in `SPACE_FooDataWriter _this` - the `SPACE_FooDataWriter` object on which the operation is operated.

in `const Foo *instance_data` - the new instance, which the application writes to or disposes of.

Return Value

`DDS_InstanceHandle_t` - Result value is the handle to the Instance, which may be used for writing and disposing of. In case of an error, a `DDS_HANDLE_NIL` handle value is returned.

Detailed Description

This operation informs the Data Distribution Service that the application will be modifying a particular instance. This operation may be invoked prior to calling any operation that modifies the instance, such as `SPACE_FooDataWriter_write`,

`SPACE_FooDataWriter_write_w_timestamp`, `SPACE_FooDataWriter_unregister_instance`, `SPACE_FooDataWriter_unregister_instance_w_timestamp`, `SPACE_FooDataWriter_dispose`, `SPACE_FooDataWriter_dispose_w_timestamp`, `SPACE_FooDataWriter_writedispose` and `SPACE_FooDataWriter_writedispose_w_timestamp`. When the application does register the instance before modifying, the Data Distribution Service will handle the instance more efficiently. It takes as a parameter (`instance_data`) an instance (to get the key value) and returns a handle that can be used in successive `DDS_DataWriter` operations. In case of an error, a `DDS_HANDLE_NIL` handle value is returned.

The explicit use of this operation is optional as the application can directly call the `SPACE_FooDataWriter_write`, `SPACE_FooDataWriter_write_w_timestamp`, `SPACE_FooDataWriter_unregister_instance`, `SPACE_FooDataWriter_unregister_instance_w_timestamp`, `SPACE_FooDataWriter_dispose`, `SPACE_FooDataWriter_dispose_w_timestamp`, `SPACE_FooDataWriter_writedispose` and `SPACE_FooDataWriter_writedispose_w_timestamp` operations and specify a `DDS_HANDLE_NIL` handle value to indicate that the sample should be examined to identify the instance.

When this operation is used, the Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Blocking

If the `DDS_HistoryQosPolicy` is set to `KEEP_ALL_HISTORY_QOS`, the `SPACE_FooDataWriter_register_instance` operation on the `DDS_DataWriter` may block if the modification would cause data to be lost because one of the limits, specified in the `DDS_ResourceLimitsQosPolicy`, to be exceeded. In case the synchronous attribute value of the `ReliabilityQosPolicy` is set to `TRUE` for communicating `DataWriters` and `DataReaders` then the `DataWriter` will wait until all synchronous `DataReaders` have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the `ReliabilityQosPolicy` configures the maximum time the `SPACE_FooDataWriter_register_instance` operation may block (either waiting for space to become available or data to be acknowledged). If `max_blocking_time` elapses before the `DDS_DataWriter` is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `SPACE_FooDataWriter_register_instance` operation will fail and returns `DDS_HANDLE_NIL`.

Sample Validation

OpenSplice DDS offers the possibility to check the sample that is passed as `instance_data` for validity. Because validity checking might reduce the overall performance, it is by default disabled. This has been done by enclosing the validity checking with conditional compiler directives like this:

```
#ifdef OSPL_BOUNDS_CHECK
    // check a specific bound.
#endif
```

i

By defining a macro called `OSPL_OSPL_BOUNDS_CHECK`, the validity checking will be included. On most compilers this macro can be defined by passing an additional command line parameter called `-DOSPL_BOUNDS_CHECK`.

Since the `SPACE_FooDataWriter_register_instance` operation merely uses the sample to determine its identity based on the uniqueness of its key values, only the keyfields will be validated against the restrictions imposed by the IDL to C language mapping, where:

- an enum may not exceed the value of its highest label
- a string (bounded or unbounded) may not be `NULL`. (Use `""` for an empty string instead)
- the length of a bounded string may not exceed the limit specified in IDL

If any of these restrictions is violated when validity checking is enabled, then the operation will fail and return a `DDS_HANDLE_NIL`. More specific information about the context of this error will be written to the error log. When validity checking is disabled, any of these violations may result in undefined behaviour.

Multiple Calls

If this operation is called for an already registered instance, it just returns the already allocated instance handle. This may be used to look up and retrieve the handle allocated to a given instance.

3.4.2.51 SPACE_FooDataWriter_register_instance_w_timestamp**Synopsis**

```
#include <Space.h>
DDS_InstanceHandle_t
SPACE_FooDataWriter_register_instance_w_timestamp
(SPACE_FooDataWriter _this,
 const Foo *instance_data,
 const DDS_Time_t *source_timestamp);
```


Description

This operation will inform the Data Distribution Service that the application will be modifying a particular instance and provides a value for the `source_timestamp` explicitly.

Parameters

in `SPACE_FooDataWriter _this` - the `SPACE_FooDataWriter` object on which the operation is operated.

in `Foo *instance_data` - the instance, which the application will write to or dispose of.

in `const DDS_Time_t *source_timestamp` - the timestamp used.

Return Value

`DDS_InstanceHandle_t` - Result value is the handle to the Instance, which must be used for writing and disposing. In case of an error, a `DDS_HANDLE_NIL` handle value is returned.

Detailed Description

This operation performs the same functions as `SPACE_FooDataWriter_register_instance` except that the application provides the value for the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Multiple Calls

If this operation is called for an already registered instance, it just returns the already allocated instance handle. This may be used to look up and retrieve the handle allocated to a given instance. The `source_timestamp` is ignored in that case.

3.4.2.52 `SPACE_FooDataWriter_set_listener` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataWriter` for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_set_listener
(
    SPACE_FooDataWriter _this,
    const struct DDS_DataWriterListener *a_listener,
    const DDS_StatusMask mask);
```

3.4.2.53 SPACE_FooDataWriter_set_qos (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataWriter for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_set_qos
        (SPACE_FooDataWriter _this,
         const DDS_DataWriterQos *qos);
```

3.4.2.54 SPACE_FooDataWriter_unregister_instance

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_unregister_instance
        (SPACE_FooDataWriter _this,
         const Foo *instance_data,
         const DDS_InstanceHandle_t handle);
```

Description

This operation informs the Data Distribution Service that the application will **not** be modifying a particular instance any more.

Parameters

in SPACE_FooDataWriter _this - the SPACE_FooDataWriter object on which the operation is operated.

*in const Foo *instance_data* - the instance to which the application was writing or disposing.

in const DDS_InstanceHandle_t handle - the handle to the instance, which has been used for writing and disposing.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_TIMEOUT.

Detailed Description

This operation informs the Data Distribution Service that the application will **not** be modifying a particular instance any more. Therefore, this operation reverses the action of `SPACE_FooDataWriter_register_instance` or `SPACE_FooDataWriter_register_instance_w_timestamp`. It should only be called on an instance that is currently registered. This operation should be called just once per instance, regardless of how many times `SPACE_FooDataWriter_register_instance` was called for that instance. This operation also indicates that the Data Distribution Service can locally remove all information regarding that instance. The application should not attempt to use the handle, previously allocated to that instance, after calling this operation.

When this operation is used, the Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Effects

If, after unregistering, the application wants to modify (write or dispose) the instance, it first has to register the instance again, or it has to use the special handle value `DDS_HANDLE_NIL`.

This operation does not indicate that the instance should be deleted (that is the purpose of `SPACE_FooDataWriter_dispose`). This operation just indicates that the `DDS_DataWriter` no longer has “anything to say” about the instance. If there is no other `DDS_DataWriter` that has registered the instance as well, then the `DDS_InstanceStateKind` in all connected `DDS_DataReaders` will be changed to `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`, provided this `DDS_InstanceStateKind` was not already set to `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`. In the last case the `DDS_InstanceStateKind` will not be effected by the `SPACE_FooDataWriter_unregister_instance` call, see also Figure 21:, *State Chart of the instance_state for a Single Instance*, on page 532.

This operation can affect the ownership of the data instance. If the `DDS_DataWriter` was the exclusive owner of the instance, calling this operation will release that ownership, meaning ownership may be transferred to another, possibly lower strength, `DDS_DataWriter`.

The operation must be called only on registered instances. Otherwise the operation returns the error `DDS_RETCODE_PRECONDITION_NOT_MET`.

Instance Handle

The `DDS_HANDLE_NIL` handle value can be used for the parameter handle. This indicates that the identity of the instance is automatically deduced from the `instance_data` (by means of the key).

If handle is any value other than `DDS_HANDLE_NIL`, then it must correspond to the value returned by `SPACE_FooDataWriter_register_instance` or `SPACE_FooDataWriter_register_instance_w_timestamp` when the instance (identified by its key) was registered. If there is no correspondence, then the result of the operation is unspecified.

The sample that is passed as `instance_data` is used to check for consistency between its key values and the supplied `instance_handle`: the sample itself will not actually be delivered to the connected `DDS_DataReaders`.

Blocking

If the `DDS_HistoryQosPolicy` is set to `DDS_KEEP_ALL_HISTORY_QOS`, then the `SPACE_FooDataWriter_unregister_instance` operation on the `DDS_DataWriter` may block if the modification would cause data to be lost because one of the limits, specified in the `DDS_ResourceLimitsQosPolicy`, to be exceeded. In case the `synchronous` attribute value of the `ReliabilityQosPolicy` is set to `TRUE` for communicating `DataWriters` and `DataReaders` then the `DataWriter` will wait until all synchronous `DataReaders` have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the `ReliabilityQosPolicy` configures the maximum time the `SPACE_FooDataWriter_unregister_instance` operation may block (either waiting for space to become available or data to be acknowledged). If `max_blocking_time` elapses before the `DDS_DataWriter` is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `SPACE_FooDataWriter_unregister_instance` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Sample Validation

OpenSplice DDS offers the possibility to check the sample that is passed as `instance_data` for validity. Because validity checking might reduce the overall performance, it is by default disabled. This has been done by enclosing the validity checking with conditional compiler directives like this:

```
#ifndef OSPL_BOUNDS_CHECK
    // check a specific bound.
#endif
```

i

By defining a macro called `OSPL_OSPL_BOUNDS_CHECK`, the validity checking will be included. On most compilers this macro can be defined by passing an additional command line parameter called `-DOSPL_BOUNDS_CHECK`.

Since the `SPACE_FooDataWriter_unregister_instance` operation merely uses the sample to check for consistency between its key values and the supplied `instance_handle`, only these keyfields will be validated against the restrictions imposed by the IDL to C language mapping:

- an enum may not exceed the value of its highest label.
- a string (bounded or unbounded) may not be NULL. (Use "" for an empty string instead).
- the length of a bounded string may not exceed the limit specified in IDL.

If any of these restrictions is violated when validity checking is enabled, the operation will fail and return a `DDS_RETCODE_BAD_PARAMETER`. More specific information about the context of this error will be written to the error log. When validity checking is disabled, any of these violations may result in undefined behaviour.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the Data Distribution Service is informed that the instance will not be modified any more.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - `handle` is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the handle has not been registered with this `SPACE_FooDataWriter`.
- `DDS_RETCODE_TIMEOUT` - either the current action overflowed the available resources as specified by the combination of the `DDS_ReliabilityQosPolicy`, `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy`, or the current action was waiting for data delivery acknowledgement by synchronous DataReaders. This caused blocking of the

SPACE_FooDataWriter_unregister_instance_w_timestamp operation, which could not be resolved before max_blocking_time of the DDS_ReliabilityQosPolicy elapsed.

3.4.2.55 SPACE_FooDataWriter_unregister_instance_w_timestamp

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_unregister_instance_w_timestamp
(SPACE_FooDataWriter _this,
 const Foo *instance_data,
 const DDS_InstanceHandle_t handle,
 const DDS_Time_t *source_timestamp);
```

Description

This operation will inform the Data Distribution Service that the application will **not** be modifying a particular instance any more and provides a value for the source_timestamp explicitly.

Parameters

in SPACE_FooDataWriter _this - the SPACE_FooDataWriter object on which the operation is operated.

*in Foo *instance_data* - the instance to which the application was writing or disposing.

in const DDS_InstanceHandle_t handle - the handle to the instance, which has been used for writing and disposing.

*in const DDS_Time_t *source_timestamp* - the timestamp used.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_TIMEOUT.

Detailed Description

This operation performs the same functions as `SPACE_FooDataWriter_unregister_instance` except that the application provides the value for the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the Data Distribution Service is informed that the instance will not be modified any more.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - handle is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the handle has not been registered with this `SPACE_FooDataWriter`.
- `DDS_RETCODE_TIMEOUT` - the current action overflowed the available resources as specified by the combination of the `DDS_ReliabilityQosPolicy`, `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy`. This caused blocking of the `SPACE_FooDataWriter_unregister_instance_w_timestamp` operation, which could not be resolved before `max_blocking_time` of the `DDS_ReliabilityQosPolicy` elapsed.

3.4.2.56 `SPACE_FooDataWriter_wait_for_acknowledgments` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataWriter` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    SPACE_FooDataWriter_wait_for_acknowledgments
        (SPACE_FooDataWriter _this,
```

```
const DDS_Duration_t *max_wait);
```

3.4.2.57 SPACE_FooDataWriter_write

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataWriter_write
        (SPACE_FooDataWriter _this,
         const Foo *instance_data,
         const DDS_InstanceHandle_t handle);
```

Description

This operation modifies the value of a data instance.

Parameters

in SPACE_FooDataWriter _this - the SPACE_FooDataWriter object on which the operation is operated.

*in const Foo *instance_data* - the data to be written.

in const DDS_InstanceHandle_t handle - the handle to the instance as supplied by SPACE_FooDataWriter_register_instance.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_TIMEOUT.

Detailed Description

This operation modifies the value of a data instance. When this operation is used, the Data Distribution Service will automatically supply the value of the *source_timestamp* that is made available to connected DDS_DataReader objects. This timestamp is important for the interpretation of the DDS_DestinationOrderQosPolicy.

As a side effect, this operation asserts liveness on the DDS_DataWriter itself and on the containing DDS_DomainParticipant.

Before writing data to an instance, the instance may be registered with the SPACE_FooDataWriter_register_instance or SPACE_FooDataWriter_register_instance_w_timestamp operation. The handle returned by one of the SPACE_FooDataWriter_register_instance

operations can be supplied to the parameter handle of the `SPACE_FooDataWriter_write` operation. However, it is also possible to supply the special `DDS_HANDLE_NIL` handle value, which means that the identity of the instance is automatically deduced from the `instance_data` (identified by the key).

Instance Handle

The `DDS_HANDLE_NIL` handle value can be used for the parameter handle. This indicates the identity of the instance is automatically deduced from the `instance_data` (by means of the key).

If handle is any value other than `DDS_HANDLE_NIL`, it must correspond to the value returned by `SPACE_FooDataWriter_register_instance` or `SPACE_FooDataWriter_register_instance_w_timestamp` when the instance (identified by its key) was registered. Passing such a registered handle helps the Data Distribution Service to process the sample more efficiently. If there is no correspondence between handle and sample, the result of the operation is unspecified.

Blocking

If the `DDS_HistoryQosPolicy` is set to `DDS_KEEP_ALL_HISTORY_QOS`, the `SPACE_FooDataWriter_write` operation on the `DDS_DataWriter` may block if the modification would cause data to be lost because one of the limits, specified in the `DDS_ResourceLimitsQosPolicy`, is exceeded. In case the synchronous attribute value of the `ReliabilityQosPolicy` is set to `TRUE` for communicating `DataWriters` and `DataReaders` then the `DataWriter` will wait until all synchronous `DataReaders` have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the `ReliabilityQosPolicy` configures the maximum time the `SPACE_FooDataWriter_write` operation may block (either waiting for space to become available or data to be acknowledged). If `max_blocking_time` elapses before the `DDS_DataWriter` is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `SPACE_FooDataWriter_write` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Sample Validation

OpenSplice DDS offers the possibility to check the sample that is passed as `instance_data` for validity. Because validity checking might reduce the overall performance, it is by default disabled. This has been done by enclosing the validity checking with conditional compiler directives like this:

```
#ifdef OSPL_BOUNDS_CHECK
    // check a specific bound.
```

```
#endif
```



By defining a macro called `OSPL_OSPL_BOUNDS_CHECK`, the validity checking will be included. On most compilers this macro can be defined by passing an additional command line parameter called `-DOSPL_BOUNDS_CHECK`.

Before the sample is accepted by the DataWriter, it is validated against the restrictions imposed by the IDL to C language mapping:

- an enum may not exceed the value of its highest label.
- a string (bounded or unbounded) may not be `NULL`. (Use `""` for an empty string instead).
- the length of a bounded string may not exceed the limit specified in IDL.
- the length of a bounded sequence may not exceed the limit specified in IDL.

If any of these restrictions is violated when validity checking is enabled, the operation will fail and return a `DDS_RETCODE_BAD_PARAMETER`. More specific information about the context of this error will be written to the error log. When validity checking is disabled, any of these violations may result in undefined behaviour.



Be aware that it is not possible for the middleware to determine whether a union is correctly initialized, since according to the IDL-C language mapping a union just returns its current contents in the format of the requested branch without performing any checks. It is therefore the responsibility of the application programmer to make sure that the requested branch actually corresponds to the currently active branch. Not doing so may result in undefined behaviour as well.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the value of a data instance is modified.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - handle is not a valid handle or instance_data is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.

- *DDS_RETCODE_PRECONDITION_NOT_MET* - the handle has not been registered with this *SPACE_FooDataWriter*.
- *DDS_RETCODE_TIMEOUT* - either the current action overflowed the available resources as specified by the combination of the *DDS_ReliabilityQosPolicy*, *DDS_HistoryQosPolicy* and *DDS_ResourceLimitsQosPolicy*, or the current action was waiting for data delivery acknowledgement by synchronous *DataReaders*. This caused blocking of the *SPACE_FooDataWriter_write* operation, which could not be resolved before *max_blocking_time* of the *DDS_ReliabilityQosPolicy* elapsed.

3.4.2.58 *SPACE_FooDataWriter_write_w_timestamp*

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataWriter_write_w_timestamp
(SPACE_FooDataWriter _this,
 const Foo *instance_data,
 const DDS_InstanceHandle_t handle,
 const DDS_Time_t *source_timestamp);
```

Description

This operation modifies the value of a data instance and provides a value for the *source_timestamp* explicitly.

Parameters

in SPACE_FooDataWriter _this - the *SPACE_FooDataWriter* object on which the operation is operated.

*in const Foo *instance_data* - the data to be written.

in const DDS_InstanceHandle_t handle - the handle to the instance as supplied by *SPACE_FooDataWriter_register_instance*.

*in const DDS_Time_t *source_timestamp* - the timestamp used.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES*, *DDS_RETCODE_NOT_ENABLED*, *DDS_RETCODE_PRECONDITION_NOT_MET* or *DDS_RETCODE_TIMEOUT*.

Detailed Description

This operation performs the same functions as `SPACE_FooDataWriter_write` except that the application provides the value for the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the value of a data instance is modified.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - handle is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the `instance_data` does not correspond to the handle that should have been obtained from this `SPACE_FooDataWriter`.
- `DDS_RETCODE_TIMEOUT` - either the current action overflowed the available resources as specified by the combination of the `DDS_ReliabilityQosPolicy`, `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy`, or the current action was waiting for data delivery acknowledgement by synchronous `DataReaders`. This caused blocking of the `SPACE_FooDataWriter_register_instance_w_timestamp` operation, which could not be resolved before `max_blocking_time` of the `DDS_ReliabilityQosPolicy` elapsed.

3.4.2.59 `SPACE_FooDataWriter_writedispose`

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataWriter_writedispose
        (SPACE_FooDataWriter _this,
         const Foo *instance_data,
         const DDS_InstanceHandle_t handle);
```

Description

This operation modifies and disposes a data instance.

Parameters

in `SPACE_FooDataWriter _this` - the `SPACE_FooDataWriter` object on which the operation is operated.

in `const Foo *instance_data` - the data to be written and disposed.

in `const DDS_InstanceHandle_t instance` - the handle to the instance as supplied by `SPACE_FooDataWriter_register_instance`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED`, `DDS_RETCODE_PRECONDITION_NOT_MET` or `DDS_RETCODE_TIMEOUT`.

Detailed Description

This operation requests the Data Distribution Service to modify the instance and mark it for deletion. Copies of the instance and its corresponding samples, which are stored in every connected `DDS_DataReader` and, dependent on the `QoSPolicy` settings, also in the Transient and Persistent stores, will be modified and marked for deletion by setting their `DDS_InstanceStateKind` to `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`.

When this operation is used, the Data Distribution Service will automatically supply the value of the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQoSPolicy`.

As a side effect, this operation asserts liveness on the `DDS_DataWriter` itself and on the containing `DDS_DomainParticipant`.

Effects on DataReaders

Actual deletion of the instance administration in a connected `DDS_DataReader` will be postponed until the following conditions have been met:

- the instance must be unregistered (either implicitly or explicitly) by all connected `DDS_DataWriters` that have previously registered it.

- A `DDS_DataWriter` can register an instance explicitly by using one of the special operations `SPACE_FooDataWriter_register_instance` or `SPACE_FooDataWriter_register_instance_w_timestamp`.
- A `DDS_DataWriter` can register an instance implicitly by using the special constant `DDS_HANDLE_NIL` in any of the other `DDS_DataWriter` operations.
- A `DDS_DataWriter` can unregister an instance explicitly by using one of the special operations `SPACE_FooDataWriter_unregister_instance` or `SPACE_FooDataWriter_unregister_instance_w_timestamp`.
- A `DDS_DataWriter` will unregister all its contained instances implicitly when it is deleted.
- When a `DDS_DataReader` detects a loss of liveliness in one of its connected `DDS_DataWriters`, it will consider all instances registered by that `DDS_DataWriter` as being implicitly unregistered.
- **and** the application must have consumed all samples belonging to the instance, either implicitly or explicitly.
 - An application can consume samples explicitly by invoking the `SPACE_FooDataReader_take` operation, or one of its variants.
 - The `DDS_DataReader` can consume disposed samples implicitly when the `autopurge_disposed_samples_delay` of the `DDS_ReaderDataLifecycleQosPolicy` has expired.

The `DDS_DataReader` may also remove instances that haven't been disposed first: this happens when the `autopurge_nowriter_samples_delay` of the `DDS_ReaderDataLifecycleQosPolicy` has expired after the instance is considered unregistered by all connected `DDS_DataWriters` (i.e. when it has a `DDS_InstanceStateKind` of `DDS_NOT_ALIVE_NO_WRITERS`). See also Section 3.1.3.15, *DDS_ReaderDataLifecycleQosPolicy*, on page 92.

Effects on Transient/Persistent Stores

Actual deletion of the instance administration in the connected Transient and Persistent stores will be postponed until the following conditions have been met:

- the instance must be unregistered (either implicitly or explicitly) by all connected `DDS_DataWriters` that have previously registered it. (See above.)
- **and** the period of time specified by the `service_cleanup_delay` attribute in the `DDS_DurabilityServiceQosPolicy` on the `DDS_Topic` must have elapsed after the instance is considered unregistered by all connected `DDS_DataWriters`.

See also Section 3.1.3.4, *DDS_DurabilityServiceQosPolicy*, on page 71.

Instance Handle

The `DDS_HANDLE_NIL` handle value can be used for the parameter handle. This indicates the identity of the instance is automatically deduced from the `instance_data` (by means of the key).

If handle is any value other than `DDS_HANDLE_NIL`, it must correspond to the value returned by `SPACE_FooDataWriter_register_instance` or `SPACE_FooDataWriter_register_instance_w_timestamp` when the instance (identified by its key) was registered. Passing such a registered handle helps the Data Distribution Service to process the sample more efficiently. If there is no correspondence between handle and sample, the result of the operation is unspecified.

The sample that is passed as `instance_data` will actually be delivered to the connected `DDS_DataReaders`, but will immediately be marked for deletion.

Blocking

If the `DDS_HistoryQosPolicy` is set to `DDS_KEEP_ALL_HISTORY_QOS`, the `SPACE_FooDataWriter_writedispose` operation on the `DDS_DataWriter` may block if the modification would cause data to be lost because one of the limits, specified in the `DDS_ResourceLimitsQosPolicy`, to be exceeded. In case the synchronous attribute value of the `ReliabilityQosPolicy` is set to `TRUE` for communicating `DataWriters` and `DataReaders` then the `DataWriter` will wait until all synchronous `DataReaders` have acknowledged the data. Under these circumstances, the `max_blocking_time` attribute of the `ReliabilityQosPolicy` configures the maximum time the `SPACE_FooDataWriter_writedispose` operation may block (either waiting for space to become available or data to be acknowledged). If `max_blocking_time` elapses before the `DDS_DataWriter` is able to store the modification without exceeding the limits and all expected acknowledgements are received, the `SPACE_FooDataWriter_writedispose` operation will fail and returns `DDS_RETCODE_TIMEOUT`.

Sample Validation

OpenSplice DDS offers the possibility to check the sample that is passed as `instance_data` for validity. Because validity checking might reduce the overall performance, it is by default disabled. This has been done by enclosing the validity checking with conditional compiler directives like this:

```
#ifdef OSPL_BOUNDS_CHECK
    // check a specific bound.
#endif
```



By defining a macro called `OSPL_OSPL_BOUNDS_CHECK`, the validity checking will be included. On most compilers this macro can be defined by passing an additional command line parameter called `-DOSPL_BOUNDS_CHECK`.

Before the sample is accepted by the DataWriter, it is validated against the restrictions imposed by the IDL to C language mapping, where:

- an enum may not exceed the value of its highest label
- a string (bounded or unbounded) may not be `NULL`. (Use `""` for an empty string instead)
- the length of a bounded string may not exceed the limit specified in IDL
- the length of a bounded sequence may not exceed the limit specified in IDL

If any of these restrictions is violated when validity checking is enabled, the operation will fail and return a `DDS_RETCODE_BAD_PARAMETER`. More specific information about the context of this error will be written to the error log. When validity checking is disabled, any of these violations may result in undefined behaviour.



Be aware that it is not possible for the middleware to determine whether a union is correctly initialized, since according to the IDL-C language mapping a union just returns its current contents in the format of the requested branch without performing any checks. It is therefore the responsibility of the application programmer to make sure that the requested branch actually corresponds to the currently active branch. Not doing so may result in undefined behaviour as well.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the Data Distribution Service has modified the instance and marked it for deletion.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - `instance_handle` is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.

- *DDS_RETCODE_PRECONDITION_NOT_MET* - the *instance_handle* has not been registered with this *SPACE_FooDataWriter*.
- *DDS_RETCODE_TIMEOUT* - either the current action overflowed the available resources as specified by the combination of the *DDS_ReliabilityQosPolicy*, *DDS_HistoryQosPolicy* and *DDS_ResourceLimitsQosPolicy*, or the current action was waiting for data delivery acknowledgement by synchronous *DataReaders*. This caused blocking of the *SPACE_FooDataWriter_writedispose* operation, which could not be resolved before *max_blocking_time* of the *DDS_ReliabilityQosPolicy* elapsed.

3.4.2.60 *SPACE_FooDataWriter_writedispose_w_timestamp*

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataWriter_writedispose_w_timestamp
        (SPACE_FooDataWriter _this,
         const Foo *instance_data,
         const DDS_InstanceHandle_t handle,
         const DDS_Time_t *source_timestamp);
```

Description

This operation requests the Data Distribution Service to modify the instance and mark it for deletion, and provides a value for the *source_timestamp* explicitly.

Parameters

in SPACE_FooDataWriter _this - the *SPACE_FooDataWriter* object on which the operation is operated.

*in const Foo *instance_data* - the data to be written and disposed.

in const DDS_InstanceHandle_t handle - the handle to the instance as supplied by *SPACE_FooDataWriter_register_instance*.

*in const DDS_Time_t *source_timestamp* - the timestamp used.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES*, *DDS_RETCODE_NOT_ENABLED*, *DDS_RETCODE_PRECONDITION_NOT_MET* or *DDS_RETCODE_TIMEOUT*.

Detailed Description

This operation performs the same functions as `SPACE_FooDataWriter_writedispose` except that the application provides the value for the `source_timestamp` that is made available to connected `DDS_DataReader` objects. This timestamp is important for the interpretation of the `DDS_DestinationOrderQosPolicy`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the Data Distribution Service has modified the instance and marked it for deletion.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - handle is not a valid handle or `instance_data` is not a valid sample.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataWriter` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataWriter` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - the handle has not been registered with this `SPACE_FooDataWriter`.
- `DDS_RETCODE_TIMEOUT` - either the current action overflowed the available resources as specified by the combination of the `DDS_ReliabilityQosPolicy`, `DDS_HistoryQosPolicy` and `DDS_ResourceLimitsQosPolicy`, or the current action was waiting for data delivery acknowledgement by synchronous `DataReaders`. This caused blocking of the `SPACE_FooDataWriter_writedispose_w_timestamp` operation, which could not be resolved before `max_blocking_time` of the `DDS_ReliabilityQosPolicy` elapsed.

3.4.3 DDS_PublisherListener interface

Since a `DDS_Publisher` is a `DDS_Entity`, it has the ability to have a `Listener` associated with it. In this case, the associated `Listener` should be of type `DDS_PublisherListener`. This interface must be implemented by the application. A user-defined class must be provided by the application which must

extend from the `DDS_PublisherListener` class. **All** `DDS_PublisherListener` operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.



All operations for this interface must be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.

The `DDS_PublisherListener` provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a `QosPolicy` setting, etc. The `DDS_PublisherListener` is related to changes in communication status.

The interface description of this class is as follows:

```
/*
 * interface DDS_PublisherListener
 */
/*
 * inherited from DDS_DataWriterListener
 */
/* void
 *     DDS_PublisherListener_on_offered_deadline_missed
 *     (void *listener_data,
 *      DDS_DataWriter writer,
 *      const DDS_OfferedDeadlineMissedStatus *status);
 */
/* void
 *     DDS_PublisherListener_on_offered_incompatible_qos
 *     (void *listener_data,
 *      DDS_DataWriter writer,
 *      const DDS_OfferedIncompatibleQosStatus *status);
 */

/* void
 *     DDS_PublisherListener_on_liveliness_lost
 *     (void *listener_data,
 *      DDS_DataWriter writer,
 *      const DDS_LivelinessLostStatus *status);
 */

/* void
 *     DDS_PublisherListener_on_publication_matched
 *     (void *listener_data,
 *      DDS_DataWriter writer,
 *      const DDS_PublicationMatchedStatus *status);
 */
/*
 * implemented API operations
```

```

*/
    DDS_PublisherListener
    DDS_PublisherListener__alloc
    (void);

```

The next paragraphs list all `DDS_PublisherListener` operations. Since these operations are all inherited, they are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.4.3.1 `DDS_PublisherListener__alloc`

Synopsis

```

#include <dds_dcps.h>
DDS_PublisherListener
    DDS_PublisherListener__alloc
    (void);

```

Description

This operation creates a new `DDS_PublisherListener`.

Parameters

<none>

Return Value

`DDS_PublisherListener` - the handle to the newly created `DDS_PublisherListener`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_PublisherListener`. The `DDS_PublisherListener` must be created using this operation. In other words, the application is not allowed to declare an object of type `DDS_PublisherListener`. When the application wants to release the `DDS_PublisherListener` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the `DDS_PublisherListener`, a `DDS_OBJECT_NIL` pointer is returned instead.

3.4.3.2 `DDS_PublisherListener_on_liveliness_lost` (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataWriterListener` for further explanation.

Synopsis

```

#include <dds_dcps.h>

```

```
void
    DDS_PublisherListener_on_liveliness_lost
    (void *listener_data,
     DDS_DataWriter writer,
     const DDS_LivelinessLostStatus *status);
```

3.4.3.3 DDS_PublisherListener_on_offered_deadline_missed (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataWriterListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_PublisherListener_on_offered_deadline_missed
    (void *listener_data,
     DDS_DataWriter writer,
     const DDS_OfferedDeadlineMissedStatus *status);
```

3.4.3.4 DDS_PublisherListener_on_offered_incompatible_qos (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataWriterListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_PublisherListener_on_offered_incompatible_qos
    (void *listener_data,
     DDS_DataWriter writer,
     const DDS_OfferedIncompatibleQosStatus *status);
```

3.4.3.5 DDS_PublisherListener_on_publication_matched (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataWriterListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_PublisherListener_on_publication_matched
    (void *listener_data,
     DDS_DataWriter writer,
     const DDS_PublicationMatchedStatus *status);
```

3.4.4 DDS_DataWriterListener interface

Since a DDS_DataWriter is a DDS_Entity, it has the ability to have a Listener associated with it. In this case, the associated Listener should be of type DDS_DataWriterListener. This interface must be implemented by the application. A user-defined class must be provided by the application which must extend from the DDS_DataWriterListener class. **All** DDS_DataWriterListener operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.



NOTE: All operations for this interface must be implemented in the user-defined class; it is up to the application whether an operation is empty or contains some functionality.

The DDS_DataWriterListener provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a QosPolicy setting, etc. The DDS_DataWriterListener is related to changes in communication status.

The interface description of this class is as follows:

```
/*
 * interface DDS_DataWriterListener
 */
/*
 * abstract external operations
 */
void
    DDS_DataWriterListener_on_offered_deadline_missed
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_OfferedDeadlineMissedStatus *status);

void
    DDS_DataWriterListener_on_offered_incompatible_qos
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_OfferedIncompatibleQosStatus *status);

void
    DDS_DataWriterListener_on_liveliness_lost
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_LivelinessLostStatus *status);

void
```

```

        DDS_DataWriterListener_on_publication_matched
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_PublicationMatchedStatus *status);
/*
 * implemented API operations
 */
DDS_DataWriterListener
DDS_DataWriterListener__alloc
(void);

```

The next paragraphs describe the usage of all `DDS_DataWriterListener` operations. These abstract operations are fully described because they must be implemented by the application.

3.4.4.1 `DDS_DataWriterListener__alloc`

Synopsis

```

#include <dds_dcps.h>
DDS_DataWriterListener
DDS_DataWriterListener__alloc
(void);

```

Description

This operation creates a new `DDS_DataWriterListener`.

Parameters

<none>

Return Value

DDS_DataWriterListener - the handle to the newly created `DDS_DataWriterListener`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_DataWriterListener`. The `DDS_DataWriterListener` must be created using this operation. In other words, the application is not allowed to declare an object of type `DDS_DataWriterListener`. When the application wants to release the `DDS_DataWriterListener` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the `DDS_DataWriterListener`, a `DDS_OBJECT_NIL` pointer is returned instead.

3.4.4.2 DDS_DataWriterListener_on_liveliness_lost (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataWriterListener_on_liveliness_lost
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_LivelinessLostStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the `DDS_LivelinessLostStatus` changes.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataWriter writer - contain a pointer to the `DDS_DataWriter` on which the `DDS_LivelinessLostStatus` has changed (this is an input to the application).

*in const DDS_LivelinessLostStatus *status* - contain the `DDS_LivelinessLostStatus` struct (this is an input to the application).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the `DDS_LivelinessLostStatus` changes. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant `DDS_DataWriterListener` is installed and enabled for the liveliness lost status. The liveliness lost status will change when the liveliness that the `DDS_DataWriter` has committed through its `DDS_LivelinessQosPolicy` was not respected. In other words, the `DDS_DataWriter` failed to actively signal its liveliness within the offered liveliness period. As a result, the `DDS_DataReader` objects will consider the `DDS_DataWriter` as no longer “alive”.

The Data Distribution Service will call the `DDS_DataWriterListener` operation with a parameter `writer`, which will contain a pointer to the `DDS_DataWriter` on which the conflict occurred and a parameter `status`, which will contain the `DDS_LivelinessLostStatus` struct.

3.4.4.3 DDS_DataWriterListener_on_offered_deadline_missed (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataWriterListener_on_offered_deadline_missed
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_OfferedDeadlineMissedStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the DDS_OfferedDeadlineMissedStatus changes.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataWriter writer - contains a pointer to the DDS_DataWriter on which the DDS_OfferedDeadlineMissedStatus has changed (this is an input to the application).

*in const DDS_OfferedDeadlineMissedStatus *status* - contains the DDS_OfferedDeadlineMissedStatus struct (this is an input to the application).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the DDS_OfferedDeadlineMissedStatus changes. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant DDS_DataWriterListener is installed and enabled for the offered deadline missed status. The offered deadline missed status will change when the deadline that the DDS_DataWriter has committed through its DDS_DeadlineQosPolicy was not respected for a specific instance.

The Data Distribution Service will call the DDS_DataWriterListener operation with a parameter *writer*, which will contain a pointer to the DDS_DataWriter on which the conflict occurred and a parameter *status*, which will contain the DDS_OfferedDeadlineMissedStatus struct.

3.4.4.4 DDS_DataWriterListener_on_offered_incompatible_qos (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataWriterListener_on_offered_incompatible_qos
        (void *listener_data,
         DDS_DataWriter writer,
         const DDS_OfferedIncompatibleQosStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the DDS_OFFERED_INCOMPATIBLE_QOS_STATUS changes.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataWriter writer - contain a pointer to the DDS_DataWriter on which the DDS_OFFERED_INCOMPATIBLE_QOS_STATUS has changed (this is an input to the application).

*in const DDS_OfferedIncompatibleQosStatus *status* - contain the DDS_OfferedIncompatibleQosStatus struct (this is an input to the application).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the DDS_OFFERED_INCOMPATIBLE_QOS_STATUS changes. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant DDS_DataWriterListener is installed and enabled for the DDS_OFFERED_INCOMPATIBLE_QOS_STATUS. The incompatible Qos status will change when a DDS_DataReader object has been discovered by the DDS_DataWriter with the same DDS_Topic and a requested DDS_DataReaderQos that was incompatible with the one offered by the DDS_DataWriter.

The Data Distribution Service will call the `DDS_DataWriterListener` operation with a parameter `writer`, which will contain a pointer to the `DDS_DataWriter` on which the conflict occurred and a parameter `status`, which will contain the `DDS_OfferedIncompatibleQosStatus` struct.

3.4.4.5 `DDS_DataWriterListener_on_publication_matched` (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataWriterListener_on_publication_matched
        (DDS_DataWritervoid *listener_data,
         DDS_DataWriter writer,
         const DDS_PublicationMatchedStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when a new match has been discovered for the current publication, or when an existing match has ceased to exist.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataWriter writer - contains a pointer to the `DDS_DataWriter` for which a match has been discovered (this is an input to the application provided by the Data Distribution Service).

*in const DDS_PublicationMatchedStatus *status* - contains the `DDS_PublicationMatchedStatus` struct (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation must be implemented by the application and is called by the Data Distribution Service when a new match has been discovered for the current publication, or when an existing match has ceased to exist. Usually this means that a new `DataReader` that matches the Topic and that has compatible Qos as the current `DDS_DataWriter` has either been discovered, or that a previously discovered `DataReader` has ceased to be matched to the current `DDS_DataWriter`. A `DataReader` may cease to match when it gets deleted, when it changes its Qos to a value that is incompatible with the current `DDS_DataWriter` or when either the

DDS_DataWriter or the DataReader has chosen to put its matching counterpart on its ignore-list using the DDS_DomainParticipant_ignore_subscription or DDS_DomainParticipant_ignore_publication operations.

The implementation of this Listener operation may be left empty when this functionality is not needed: it will only be called when the relevant `DDS_DataWriterListener` is installed and enabled for the `DDS_PUBLICATION_MATCHED_STATUS`.

The Data Distribution Service will provide a pointer to the DDS_DataWriter in the parameter `writer` and the DDS_PublicationMatchedException struct in the parameter `status` for use by the application.

3.5 Subscription Module

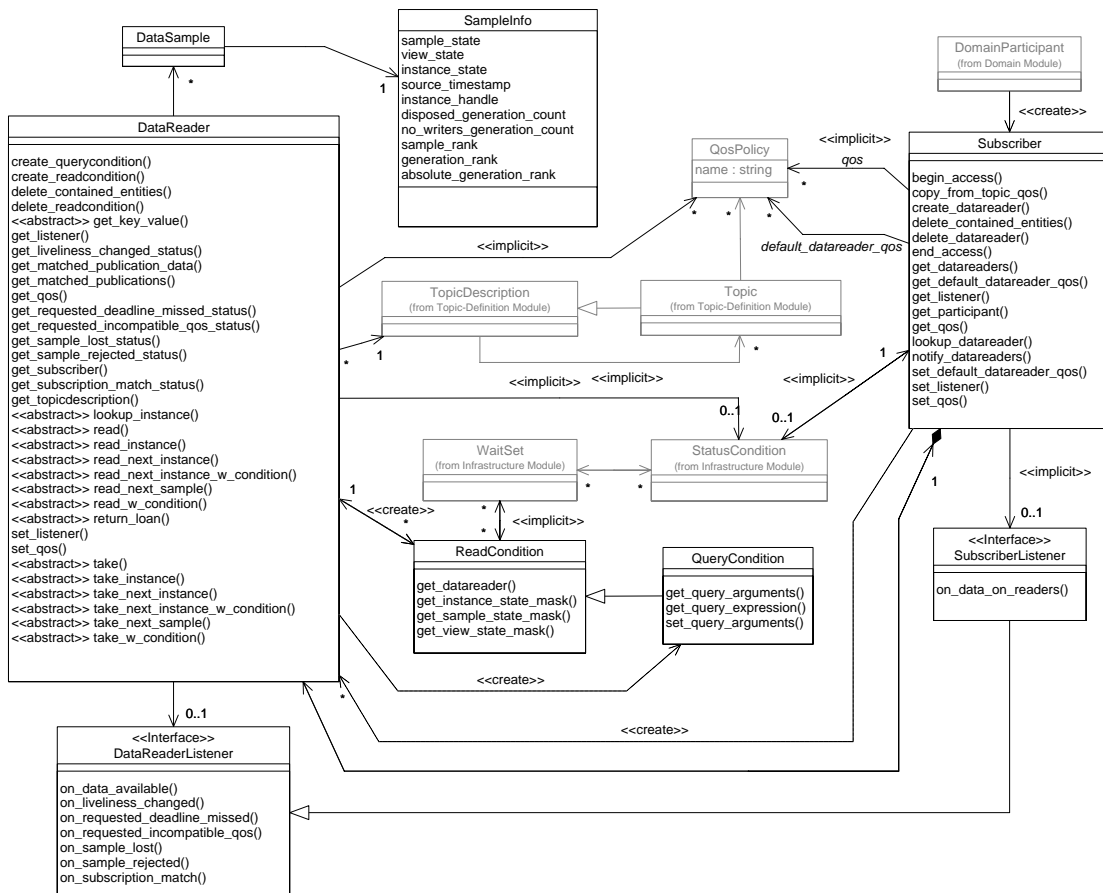


Figure 19 The DCPS Subscription Module's Class Model

This module contains the following classes:

- `DDS_Subscriber`
- Subscription type specific classes
- `DDS_DataSample`
- `DDS_SampleInfo` (struct)
- `DDS_SubscriberListener` (interface)
- `DDS_DataReaderListener` (interface)
- `DDS_ReadCondition`
- `DDS_QueryCondition`

“Subscription type specific classes” contains the generic class and the generated data type specific classes. For each data type, a data type specific class `<NameSpace>_<type>DataReader` is generated (based on IDL) by calling the pre-processor.

For instance, for the fictional data type `Foo` (this also applies to other types), defined in the module `SPACE`; “Subscription type specific classes” contains the following classes:

- `DDS_DataReader` (abstract)
- `SPACE_FooDataReader`

A `DDS_Subscriber` is an object responsible for receiving published data and making it available (according to the `DDS_SubscriberQos`) to the application. It may receive and dispatch `DDS_Topic` with data of different specified data types. To access the received data, the application must use a typed `DDS_DataReader` attached to the `DDS_Subscriber`. Thus, a subscription is defined by the association of a `DDS_DataReader` with a `DDS_Subscriber`. This association expresses the intent of the application to subscribe to the data described by the `DDS_DataReader` in the context provided by the `DDS_Subscriber`.

3.5.1 Class `DDS_Subscriber`

A `DDS_Subscriber` is the object responsible for the actual reception of the data resulting from its subscriptions.

A `DDS_Subscriber` acts on behalf of one or more `DDS_DataReader` objects that are related to it. When it receives data (from the other parts of the system), it indicates to the application that data is available through its `DDS_DataReaderListener` and by enabling related `DDS_Conditions`. The application can access the list of concerned `DDS_DataReader` objects through the operation `DDS_Subscriber_get_datareaders` and then access the data available through operations on the `DDS_DataReader`.

The interface description of this class is as follows:

```

/*
 * interface DDS_Subscriber
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *   DDS_Subscriber_get_statuscondition
 *   (DDS_Subscriber _this)
 */
/* DDS_StatusMask
 *   DDS_Subscriber_get_status_changes
 *   (DDS_Subscriber _this);
 */
/* DDS_ReturnCode_t
 *   DDS_Subscriber_enable
 *   (DDS_Subscriber _this);
 */
/*
 * implemented API operations
 */
    DDS_DataReader
        DDS_Subscriber_create_datareader
            (DDS_Subscriber _this,
             const DDS_TopicDescription a_topic,
             const DDS_DataReaderQos *qos,
             const struct DDS_DataReaderListener *a_listener,
             const DDS_StatusMask mask);

    DDS_ReturnCode_t
        DDS_Subscriber_delete_datareader
            (DDS_Subscriber _this,
             const DDS_DataReader a_datareader);

    DDS_ReturnCode_t
        DDS_Subscriber_delete_contained_entities
            (DDS_Subscriber _this);

    DDS_DataReader
        DDS_Subscriber_lookup_datareader
            (DDS_Subscriber _this,
             const DDS_char *topic_name);

    DDS_ReturnCode_t
        DDS_Subscriber_get_datareaders
            (DDS_Subscriber _this,
             DDS_DataReaderSeq *readers,
             const DDS_SampleStateMask sample_states,

```

```

        const DDS_ViewStateMask view_states,
        const DDS_InstanceStateMask instance_states);

DDS_ReturnCode_t
    DDS_Subscriber_notify_datareaders
        (DDS_Subscriber _this);

DDS_ReturnCode_t
    DDS_Subscriber_set_qos
        (DDS_Subscriber _this,
         const DDS_SubscriberQos *qos);

DDS_ReturnCode_t
    DDS_Subscriber_get_qos
        (DDS_Subscriber _this,
         DDS_SubscriberQos *qos);
DDS_ReturnCode_t
    DDS_Subscriber_set_listener
        (DDS_Subscriber _this,
         const struct DDS_SubscriberListener *a_listener,
         const DDS_StatusMask mask);

struct DDS_SubscriberListener
    DDS_Subscriber_get_listener
        (DDS_Subscriber _this);

DDS_ReturnCode_t
    DDS_Subscriber_begin_access
        (DDS_Subscriber _this);

DDS_ReturnCode_t
    DDS_Subscriber_end_access
        (DDS_Subscriber _this);

DDS_DomainParticipant
    DDS_Subscriber_get_participant
        (DDS_Subscriber _this);

DDS_ReturnCode_t
    DDS_Subscriber_set_default_datareader_qos
        (DDS_Subscriber _this,
         const DDS_DataReaderQos *qos);

DDS_ReturnCode_t
    DDS_Subscriber_get_default_datareader_qos
        (DDS_Subscriber _this,
         DDS_DataReaderQos *qos);

DDS_ReturnCode_t
    DDS_Subscriber_copy_from_topic_qos

```

```
(DDS_Subscriber _this,
    DDS_DataReaderQos *a_datareader_qos,
    const DDS_TopicQos *a_topic_qos);
```

The next paragraphs describe the usage of all DDS_Subscriber operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.5.1.1 DDS_Subscriber_begin_access

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_begin_access
        (DDS_Subscriber _this);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.1.2 DDS_Subscriber_copy_from_topic_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_copy_from_topic_qos
        (DDS_Subscriber _this,
         DDS_DataReaderQos *a_datareader_qos,
         const DDS_TopicQos *a_topic_qos);
```

Description

This operation will copy the policies in *a_topic_qos* to the corresponding policies in *a_datareader_qos*.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

*inout DDS_DataReaderQos *a_datareader_qos* - the destination DDS_DataReaderQos struct to which the QosPolicy settings will be copied.

*in const DDS_TopicQos *a_topic_qos* - the source DDS_TopicQos, which will be copied.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation will copy the QosPolicy settings in *a_topic_qos* to the corresponding QosPolicy settings in *a_datareader_qos* (replacing the values in *a_datareader_qos*, if present).

This is a “convenience” operation, useful in combination with the operations *DDS_Publisher_get_default_datawriter_qos* and *DDS_Topic_get_qos*. The operation *DDS_Subscriber_copy_from_topic_qos* can be used to merge the *DDS_DataReader* default QosPolicy settings with the corresponding ones on the *DDS_Topic*. The resulting *DDS_DataReaderQos* can then be used to create a new *DDS_DataReader*, or set its *DDS_DataReaderQos*.

This operation does not check the resulting *a_datareader_qos* for self consistency. This is because the “merged” *a_datareader_qos* may not be the final one, as the application can still modify some QosPolicy settings prior to applying the *DDS_DataReaderQos* to the *DDS_DataReader*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the QosPolicy settings have successfully been copied from the *DDS_TopicQos* to the *DDS_DataReaderQos*.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Subscriber* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.1.3 DDS_Subscriber_create_datareader

Synopsis

```
#include <dds_dcps.h>
DDS_DataReader
    DDS_Subscriber_create_datareader
        (DDS_Subscriber _this,
         const DDS_TopicDescription a_topic,
```

```
const DDS_DataReaderQos *qos,
const struct DDS_DataReaderListener *a_listener,
const DDS_StatusMask mask);
```

Description

This operation creates a DDS_DataReader with the desired QoSPolicy settings, for the desired DDS_TopicDescription and attaches the optionally specified DDS_DataWriterListener to it.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

in const DDS_TopicDescription a_topic - a pointer to the DDS_TopicDescription for which the DDS_DataReader is created. This may be a DDS_Topic, DDS_MultiTopic or DDS_ContentFilteredTopic.

*in const DDS_DataReaderQos *qos* - the struct with the QoSPolicy settings for the new DDS_DataReader, when these QoSPolicy settings are not self consistent, no DDS_DataReader is created.

*in const struct DDS_DataReaderListener *a_listener* - a pointer to the DDS_DataReaderListener instance which will be attached to the new DDS_DataReader. It is permitted to use DDS_OBJECT_NIL as the value of the listener: this behaves as a DDS_DataWriterListener whose operations perform no action.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the DDS_DataReaderListener for a certain status.

Return Value

DDS_DataReader - Return value is a pointer to the newly created DDS_DataReader. In case of an error, the DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation creates a DDS_DataReader with the desired QoSPolicy settings, for the desired DDS_TopicDescription and attaches the optionally specified DDS_DataReaderListener to it. The DDS_TopicDescription may be a DDS_Topic, DDS_MultiTopic or DDS_ContentFilteredTopic. The returned DDS_DataReader is attached (and belongs) to the DDS_Subscriber. To delete the DDS_DataReader the operation DDS_Subscriber_delete_datareader or DDS_Subscriber_delete_contained_entities must be used. If no read rights are defined for the specific topic then the creation of the DataReader will fail.

Application Data Type

The DDS_DataReader returned by this operation is an object of a derived class, specific to the data type associated with the DDS_TopicDescription. For each application-defined data type <type> there is a class <NameSpace>_<type>DataReader generated by calling the pre-processor. This data type specific class extends DDS_DataReader and contains the operations to read data of data type <type>.

Because the DDS_DataReader may read a DDS_Topic, DDS_ContentFilteredTopic or DDS_MultiTopic, the DDS_DataReader is associated with the DDS_TopicDescription. The DDS_DataWriter can only write a DDS_Topic, **not** a DDS_ContentFilteredTopic or DDS_MultiTopic, because these two are constructed at the DDS_Subscriber side.

QosPolicy

The common application pattern to construct the QosPolicy settings for the DDS_DataReader is to:

- Retrieve the QosPolicy settings on the associated DDS_TopicDescription by means of the DDS_Topic_get_qos operation on the DDS_TopicDescription
- Retrieve the default DDS_DataReaderQos by means of the DDS_Subscriber_get_default_datareader_qos operation on the DDS_Subscriber
- Combine those two QosPolicy settings and selectively modify policies as desired (DDS_Subscriber_copy_from_topic_qos)
- Use the resulting QosPolicy settings to construct the DDS_DataReader.
- In case the specified QosPolicy settings are not self consistent, no DDS_DataReader is created and the DDS_OBJECT_NIL pointer is returned.

Default QoS

The constant DDS_DATAREADER_QOS_DEFAULT can be used as parameter qos to create a DDS_DataReader with the default DDS_DataReaderQos as set in the DDS_Subscriber. The effect of using DDS_DATAREADER_QOS_DEFAULT is the same as calling the operation DDS_Subscriber_get_default_datareader_qos and using the resulting DDS_DataReaderQos to create the DDS_DataReader.

The special DDS_DATAREADER_QOS_USE_TOPIC_QOS can be used to create a DDS_DataReader with a combination of the default DDS_DataReaderQos and the DDS_TopicQos. The effect of using DDS_DATAREADER_QOS_USE_TOPIC_QOS is the same as calling the operation

DDS_Subscriber_get_default_datareader_qos and retrieving the DDS_TopicQos (by means of the operation DDS_Topic_get_qos) and then combining these two QosPolicy settings using the operation DDS_Subscriber_copy_from_topic_qos, whereby any common policy that is set on the DDS_TopicQos “overrides” the corresponding policy on the default DDS_DataReaderQos. The resulting DDS_DataReaderQos is then applied to create the DDS_DataReader.

Communication Status

For each communication status, the StatusChangedFlag flag is initially set to FALSE. It becomes TRUE whenever that communication status changes. For each communication status activated in the mask, the associated DDS_DataReaderListener operation is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The fact that the status is reset prior to calling the listener means that if the application calls the get_<status_name>_status from inside the listener it will see the status already reset.

The following statuses are applicable to the DDS_DataReaderListener:

```
DDS_REQUESTED_DEADLINE_MISSED_STATUS
DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
DDS_SAMPLE_LOST_STATUS
DDS_SAMPLE_REJECTED_STATUS
DDS_DATA_AVAILABLE_STATUS
DDS_LIVELINESS_CHANGED_STATUS
DDS_SUBSCRIPTION_MATCHED_STATUS.
```



Be aware that the DDS_SUBSCRIPTION_MATCHED_STATUS is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the NetworkingService/Discovery/enabled property in the Deployment Manual for more information about this subject.) In this case the operation will return DDS_OBJECT_NIL.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant DDS_STATUS_MASK_NONE can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant STATUS_MASK_ANY_V1_2 can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

In case a communication status is not activated in the mask of the DDS_DataReaderListener, the DDS_SubscriberListener of the containing DDS_Subscriber is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the DDS_SubscriberListener of the containing DDS_Subscriber and a DDS_DataReader specific behaviour when needed. In case the communication status is not activated in the mask of the DDS_SubscriberListener as well, the communication status will be propagated to the DDS_DomainParticipantListener of the containing DDS_DomainParticipant. In case the DDS_DomainParticipantListener is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

3.5.1.4 DDS_Subscriber_delete_contained_entities

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_delete_contained_entities
        (DDS_Subscriber _this);
```

Description

This operation deletes all the DDS_DataReader objects that were created by means of the DDS_Subscriber_create_datareader operation on the DDS_Subscriber.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation deletes all the DDS_DataReader objects that were created by means of the DDS_Subscriber_create_datareader operation on the DDS_Subscriber. In other words, it deletes all contained DDS_DataReader objects. Prior to deleting each DDS_DataReader, this operation recursively calls

the corresponding `DDS_DataReader_delete_contained_entities` operation on each `DDS_DataReader`. In other words, all `DDS_DataReader` objects in the `DDS_Subscriber` are deleted, including the `DDS_QueryCondition` and `DDS_ReadCondition` objects contained by the `DDS_DataReader`.



NOTE: The operation will return `DDS_PRECONDITION_NOT_MET` if the any of the contained entities is in a state where it cannot be deleted. This will occur, for example, if a contained `DDS_DataReader` cannot be deleted because the application has called a `read` or `take` operation and has not called the corresponding `return_loan` operation to return the loaned samples. In such cases, the operation does not roll back any entity deletions performed prior to the detection of the problem.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the contained `DDS_Entity` objects are deleted and the application may delete the `DDS_Subscriber`.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Subscriber` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one or more of the contained entities are in a state where they cannot be deleted.

3.5.1.5 `DDS_Subscriber_delete_datareader`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_delete_datareader
        (DDS_Subscriber _this,
         const DDS_DataReader a_datareader);
```

Description

This operation deletes a `DDS_DataReader` that belongs to the `DDS_Subscriber`.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

in const DDS_DataReader a_datareader - a pointer to the DDS_DataReader, which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation deletes a DDS_DataReader that belongs to the DDS_Subscriber. When the operation is called on a different DDS_Subscriber as used when the DDS_DataReader was created, the operation has no effect and returns DDS_RETCODE_PRECONDITION_NOT_MET. The deletion of the DDS_DataReader is not allowed if there are any DDS_ReadCondition or DDS_QueryCondition objects that are attached to the DDS_DataReader, or when the DDS_DataReader still contains unreturned loans. In those cases the operation also returns DDS_RETCODE_PRECONDITION_NOT_MET.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_DataReader is deleted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *a_datareader* is not a valid DDS_DataReader.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Subscriber has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

- *DDS_RETCODE_PRECONDITION_NOT_MET* - the operation is called on a different *DDS_Subscriber* as used when the *DDS_DataReader* was created, the *DDS_DataReader* contains one or more *DDS_ReadCondition* or *DDS_QueryCondition* objects or the *DDS_DataReader* still contains unreturned loans.

3.5.1.6 DDS_Subscriber_enable (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_enable
        (DDS_Subscriber _this);
```

3.5.1.7 DDS_Subscriber_end_access

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_end_access
        (DDS_Subscriber _this);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.1.8 DDS_Subscriber_get_datareaders

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_get_datareaders
        (DDS_Subscriber _this,
         DDS_DataReaderSeq *readers,
         const DDS_SampleStateMask sample_states,
         const DDS_ViewStateMask view_states,
         const DDS_InstanceStateMask instance_states);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.1.9 DDS_Subscriber_get_default_datareader_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_get_default_datareader_qos
        (DDS_Subscriber _this,
```



```
DDS_DataReaderQos *qos);
```

Description

This operation gets the default QosPolicy settings of the DDS_DataReader.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

*inout DDS_DataReaderQos *qos* - a pointer to the DDS_DataReaderQos struct (provided by the application) in which the default QosPolicy settings for the DDS_DataReader are written.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation gets the default QosPolicy settings of the DDS_DataReader (that is the DDS_DataReaderQos) which is used for newly created DDS_DataReader objects, in case the constant DDS_DATAREADER_QOS_DEFAULT is used. The default DDS_DataReaderQos is only used when the constant is supplied as parameter *qos* to specify the DDS_DataReaderQos in the DDS_Subscriber_create_datareader operation. The application must provide the DDS_DataReaderQos struct in which the QosPolicy settings can be stored and pass the *qos* pointer to the operation. The operation writes the default QosPolicy settings to the struct pointed to by *qos*. Any settings in the struct are overwritten.

The values retrieved by this operation match the values specified on the last successful call to DDS_Subscriber_set_default_datareader_qos, or, if the call was never made, the default values as specified for each QosPolicy setting as defined in Table 5 on page 59.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the default DDS_DataReader QosPolicy settings of this DDS_Subscriber have successfully been copied into the specified DDS_DataReaderQos parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Subscriber* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.1.10 DDS_Subscriber_get_listener

Synopsis

```
#include <dds_dcps.h>
struct DDS_SubscriberListener
    DDS_Subscriber_get_listener
        (DDS_Subscriber _this);
```

Description

This operation allows access to a *DDS_SubscriberListener*.

Parameters

in DDS_Subscriber _this - the *DDS_Subscriber* object on which the operation is operated.

Return Value

struct DDS_SubscriberListener - result is a pointer to the *DDS_SubscriberListener* attached to the *DDS_Subscriber*.

Detailed Description

This operation allows access to a *DDS_SubscriberListener* attached to the *DDS_Subscriber*. When no *DDS_SubscriberListener* was attached to the *DDS_Subscriber*, the *DDS_OBJECT_NIL* pointer is returned.

3.5.1.11 DDS_Subscriber_get_participant

Synopsis

```
#include <dds_dcps.h>
DDS_DomainParticipant
    DDS_Subscriber_get_participant
        (DDS_Subscriber _this);
```

Description

This operation returns the *DDS_DomainParticipant* associated with the *DDS_Subscriber* or the *DDS_OBJECT_NIL* pointer.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

Return Value

DDS_DomainParticipant - a pointer to the DDS_DomainParticipant associated with the DDS_Subscriber or the DDS_OBJECT_NIL pointer.

Detailed Description

This operation returns the DDS_DomainParticipant associated with the DDS_Subscriber. Note that there is exactly one DDS_DomainParticipant associated with each DDS_Subscriber. When the DDS_Subscriber was already deleted (there is no associated DDS_DomainParticipant any more), the DDS_OBJECT_NIL pointer is returned.

3.5.1.12 DDS_Subscriber_get_qos**Synopsis**

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_get_qos
        (DDS_Subscriber _this,
         DDS_SubscriberQos *qos);
```

Description

This operation allows access to the existing set of QoS policies for a DDS_Subscriber.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

*inout DDS_SubscriberQos *qos* - a pointer to the destination DDS_SubscriberQos struct in which the QosPolicy settings will be copied.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation allows access to the existing set of QoS policies of a `DDS_Subscriber` on which this operation is used. This `DDS_SubscriberQos` is stored at the location pointed to by the `qos` parameter.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of QoS policy values applied to this `DDS_Subscriber` has successfully been copied into the specified `DDS_SubscriberQos` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Subscriber` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.5.1.13 `DDS_Subscriber_get_status_changes` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
    DDS_Subscriber_get_status_changes
        (DDS_Subscriber _this);
```

3.5.1.14 `DDS_Subscriber_get_statuscondition` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
    DDS_Subscriber_get_statuscondition
        (DDS_Subscriber _this);
```

3.5.1.15 DDS_Subscriber_lookup_datareader

Synopsis

```
#include <dds_dcps.h>
DDS_DataReader
    DDS_Subscriber_lookup_datareader
        (DDS_Subscriber _this,
         const DDS_char *topic_name);
```

Description

This operation returns a previously created DDS_DataReader belonging to the DDS_Subscriber which is attached to a DDS_Topic with the matching topic_name.

Parameters

in DDS_Subscriber _this - the DDS_Subscriber object on which the operation is operated.

*in const DDS_char *topic_name* - the name of the DDS_Topic, which is attached to the DDS_DataReader to look for.

Return Value

DDS_DataReader - Return value is a pointer to the DDS_DataReader found. When no such DDS_DataReader is found, the DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation returns a previously created DDS_DataReader belonging to the DDS_Subscriber which is attached to a DDS_Topic with the matching topic_name. When multiple DDS_DataReader objects (which satisfy the same condition) exist, this operation will return one of them. It is not specified which one.

This operation may be used on the built-in DDS_Subscriber, which returns the built-in DDS_DataReader objects for the built-in DDS_Topics.

3.5.1.16 DDS_Subscriber_notify_datareaders

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_notify_datareaders
        (DDS_Subscriber _this);
```

Description

This operation invokes the `DDS_DataReaderListener_on_data_available` operation on `DDS_DataReaderListener` objects which are attached to the contained `DDS_DataReader` entities having new, available data.

Parameters

in `DDS_Subscriber_this` - the `DDS_Subscriber` object on which the operation is operated.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`,
`DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED`
or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation invokes the `DDS_DataReaderListener_on_data_available` operation on the `DDS_DataReaderListener` objects attached to contained `DDS_DataReader` entities that have received information, but which have not yet been processed by those `DDS_DataReaders`.

The `DDS_Subscriber_notify_datareaders` operation ignores the bit mask value of the individual `DDS_DataReaderListener` objects, even when the `DDS_DATA_AVAILABLE_STATUS` bit has not been set on a `DDS_DataReader` that which has new, available data. The `DDS_DataReaderListener_on_data_available` operation will still be invoked, when the `DATA_AVAILABLE_STATUS` bit has not been set on a `DataReader`, but will not propagate to the `DDS_DomainParticipantListener`.

When the `DDS_DataReader` has attached a `NULL` listener, the event will be consumed and will not propagate to the `DDS_DomainParticipantListener`. (Remember that a `NULL` listener is regarded as a listener that handles all its events as a `NOOP`).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - all appropriate listeners have been invoked
- `DDS_RETCODE_ERROR` - an internal error has occurred
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object

- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Subscriber* has already been deleted
- *DDS_RETCODE_OUT_OF_RESOURCES* - there are insufficient Data Distribution Service resources to complete this operation

3.5.1.17 *DDS_Subscriber_set_default_datareader_qos*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_set_default_datareader_qos
        (DDS_Subscriber _this,
         const DDS_DataReaderQos *qos);
```

Description

This operation sets the default *DDS_DataReaderQos* of the *DDS_DataReader*.

Parameters

in DDS_Subscriber _this - the *DDS_Subscriber* object on which the operation is operated.

*in const DDS_DataReaderQos *qos* - the *DDS_DataReaderQos* struct, which contains the new default *QosPolicy* settings for the newly created *DDS_DataReaders*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_INCONSISTENT_POLICY*.

Detailed Description

This operation sets the default *DDS_DataReaderQos* of the *DDS_DataReader* (that is the struct with the *QosPolicy* settings). This *QosPolicy* is used for newly created *DDS_DataReader* objects in case the constant *DDS_DATAREADER_QOS_DEFAULT* is used as parameter *qos* to specify the *DDS_DataReaderQos* in the *DDS_Subscriber_create_datareader* operation. This operation checks if the *DDS_DataReaderQos* is self consistent. If it is not, the operation has no effect and returns *DDS_RETCODE_INCONSISTENT_POLICY*.

The values set by this operation are returned by *DDS_Subscriber_get_default_datareader_qos*.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the new default `DDS_DataReaderQos` is set.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `qos` is not a valid `DDS_DataReaderQos`. It contains a `QosPolicy` setting with an invalid `DDS_Duration_t` value, an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- `DDS_RETCODE_UNSUPPORTED` - one or more of the selected `QosPolicy` values are currently not supported by OpenSplice.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_Subscriber` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_INCONSISTENT_POLICY` - the parameter `qos` contains conflicting `QosPolicy` settings, e.g. a history depth that is higher than the specified resource limits.

3.5.1.18 `DDS_Subscriber_set_listener`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_set_listener
        (DDS_Subscriber _this,
         const struct DDS_SubscriberListener *a_listener,
         const DDS_StatusMask mask);
```

Description

This operation attaches a `DDS_SubscriberListener` to the `DDS_Subscriber`.

Parameters

in `DDS_Subscriber _this` - the `DDS_Subscriber` object on which the operation is operated.

in `const struct DDS_SubscriberListener *a_listener` - a pointer to the `DDS_SubscriberListener` instance, which will be attached to the `DDS_Subscriber`.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the DDS_SubscriberListener for a certain status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:

DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation attaches a DDS_SubscriberListener to the DDS_Subscriber. Only one DDS_SubscriberListener can be attached to each DDS_Subscriber. If a DDS_SubscriberListener was already attached, the operation will replace it with the new one. When *a_listener* is the DDS_OBJECT_NIL pointer, it represents a listener that is treated as a NOOP¹ for all statuses activated in the bitmask.

Communication Status

For each communication status, the StatusChangedFlag flag is initially set to FALSE. It becomes TRUE whenever that communication status changes. For each communication status activated in the *mask*, the associated DDS_SubscriberListener operation is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the *get_<status_name>_status* from inside the listener it will see the status already reset. An exception to this rule is the DDS_OBJECT_NIL listener, which does not reset the communication statuses for which it is invoked.

The following statuses are applicable to the DDS_SubscriberListener:

- DDS_REQUESTED_DEADLINE_MISSED_STATUS *(propagated)*
- DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS *(propagated)*
- DDS_SAMPLE_LOST_STATUS *(propagated)*
- DDS_SAMPLE_REJECTED_STATUS *(propagated)*
- DDS_DATA_AVAILABLE_STATUS *(propagated)*
- DDS_LIVELINESS_CHANGED_STATUS *(propagated)*
- DDS_SUBSCRIPTION_MATCHED_STATUS *(propagated).*
- DDS_DATA_ON_READERS_STATUS.

1. Short for **No-Operation**, an instruction that performs nothing at all.



Be aware that the `DDS_SUBSCRIPTION_MATCHED_STATUS` is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

The Data Distribution Service will trigger the most specific and relevant `Listener`. In other words, in case a communication status is also activated on the `DDS_DataReaderListener` of a contained `DDS_DataReader`, the `DDS_DataReaderListener` on that contained `DDS_DataReader` is invoked instead of the `DDS_SubscriberListener`. This means that a status change on a contained `DDS_DataReader` only invokes the `DDS_SubscriberListener` if the contained `DDS_DataReader` itself does not handle the trigger event generated by the status change.

In case a communication status is not activated in the mask of the `DDS_SubscriberListener`, the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant` and a `DDS_Subscriber` specific behaviour when needed. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

The statuses `DDS_DATA_ON_READERS_STATUS` and `DDS_DATA_AVAILABLE_STATUS` are “Read Communication Statuses” and are an exception to all other plain communication statuses: they have no corresponding status structure that can be obtained with a `get_<status_name>_status` operation and they are mutually exclusive. When new information becomes available to a `DataReader`, the Data Distribution Service will first look in an attached and activated `DDS_SubscriberListener` or `DDS_DomainParticipantListener` (in that order) for the `DDS_DATA_ON_READERS_STATUS`. In case the

DDS_DATA_ON_READERS_STATUS can not be handled, the Data Distribution Service will look in an attached and activated DDS_DataReaderListener, DDS_SubscriberListener or DDS_DomainParticipantListener for the DDS_DATA_AVAILABLE_STATUS (in that order).

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the DDS_SubscriberListener is attached.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_UNSUPPORTED* - a status was selected that cannot be supported because the infrastructure does not maintain the required connectivity information.
- *DDS_RETCODE_ALREADY_DELETED* - the DDS_Subscriber has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.1.19 DDS_Subscriber_set_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_Subscriber_set_qos
        (DDS_Subscriber _this,
         const DDS_SubscriberQos *qos);
```

Description

This operation replaces the existing set of QosPolicy settings for a DDS_Subscriber.

Parameters

- in DDS_Subscriber _this* - the DDS_Subscriber object on which the operation is operated.
- in const DDS_SubscriberQos *qos* - contain the new set of QosPolicy settings for the DDS_Subscriber.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_IMMUTABLE_POLICY or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation replaces the existing set of *QoSPolicy* settings for a *DDS_Subscriber*. The parameter *qos* must contain the *QoSPolicy* settings which is checked for self-consistency and mutability. When the application tries to change a *QoSPolicy* setting for an enabled *DDS_Subscriber*, which can only be set before the *DDS_Subscriber* is enabled, the operation will fail and a *DDS_RETCODE_IMMUTABLE_POLICY* is returned. In other words, the application must provide the presently set *QoSPolicy* settings in case of the immutable *QoSPolicy* settings. Only the mutable *QoSPolicy* settings can be changed. When *qos* contains conflicting *QoSPolicy* settings (not self-consistent), the operation will fail and a *RETCODE_INCONSISTENT_POLICY* is returned.

The set of *QoSPolicy* settings specified by the *qos* parameter are applied on top of the existing *QoS*, replacing the values of any policies previously set (provided, the operation returned *DDS_RETCODE_OK*). If one or more of the partitions in the *QoS* structure have insufficient access rights configured then the *set_qos* function will fail with a *DDS_RETCODE_PRECONDITION_NOT_MET* error code.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the new *DDS_SubscriberQos* is set.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *qos* is not a valid *DDS_SubscriberQos*. It contains a *QoSPolicy* setting with an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- *DDS_RETCODE_UNSUPPORTED* - one or more of the selected *QoSPolicy* values are currently not supported by OpenSplice.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_Subscriber* has already been deleted.

- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_IMMUTABLE_POLICY* - the parameter *qos* contains an immutable *QosPolicy* setting with a different value than set during enabling of the *DDS_Subscriber*.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - returned when insufficient access rights exist for the partition(s) listed in the *QoS* structure.

3.5.2 Subscription Type Specific Classes

“Subscription type specific classes” contains the generic class and the generated data type specific classes. For each data type, a data type specific class `<NameSpace>_<type>DataReader` is generated (based on IDL) by calling the pre-processor. In case of data type `Foo` (this also applies to other types), defined in the module `SPACE`; “Subscription type specific classes” contains the following classes:

This paragraph describes the generic `DDS_DataReader` class and the derived application type specific `<NameSpace>_<type>DataReader` classes which together implement the application subscription interface. For each application type, used as `DDS_Topic` data type, the pre-processor generates a `<NameSpace>_<type>DataReader` class from an IDL type description. The `SPACE_FooDataReader` class that would be generated by the pre-processor for a fictional type `Foo` (defined in the module `SPACE`) describes the `<NameSpace>_<type>DataReader` classes.

3.5.2.1 Class `DDS_DataReader` (abstract)

A `DDS_DataReader` allows the application:

- to declare data it wishes to receive (i.e., make a subscription)
- to access data received by the associated `DDS_Subscriber`.

A `DDS_DataReader` refers to exactly one `DDS_TopicDescription` (either a `DDS_Topic`, a `DDS_ContentFilteredTopic` or a `DDS_MultiTopic`) that identifies the samples to be read. The `DDS_DataReader` may give access to several instances of the data type, which are distinguished from each other by their key.

`DDS_DataReader` is an abstract class. It is specialized for each particular application data type. For a fictional application data type “`Foo`” the specialized class would be `SPACE_FooDataReader`.

The interface description of this class is as follows:

```
/*
 * interface DDS_DataReader
 */
/*
```

```

    * inherited from class DDS_Entity
    */
/* DDS_StatusCondition
    *     DDS_DataReader_get_statuscondition
    *     (DDS_DataReader _this);
    */
/* DDS_StatusMask
    *     DDS_DataReader_get_status_changes
    *     (DDS_DataReader _this);
    */
/* DDS_ReturnCode_t
    *     DDS_DataReader_enable
    *     (DDS_DataReader _this);
    */
/*
    * abstract operations
    * (implemented in the data type specific DDS_DataReader)
    */
/* DDS_ReturnCode_t
    *     DDS_DataReader_read
    *     (DDS_DataReader _this,
    *      DDS_sequence_<data> *data_values,
    *      DDS_SampleInfoSeq *info_seq,
    *      const DDS_long max_samples,
    *      const DDS_SampleStateMask sample_states,
    *      const DDS_ViewStateMask view_states,
    *      const DDS_InstanceStateMask instance_states);
    */
/* DDS_ReturnCode_t
    *     DDS_DataReader_take
    *     (DDS_DataReader _this,
    *      DDS_sequence_<data> *data_values,
    *      DDS_SampleInfoSeq *info_seq,
    *      const DDS_long max_samples,
    *      const DDS_SampleStateMask sample_states,
    *      const DDS_ViewStateMask view_states,
    *      const DDS_InstanceStateMask instance_states);
    */
/* DDS_ReturnCode_t
    *     DDS_DataReader_read_w_condition
    *     (DDS_DataReader _this,
    *      DDS_sequence_<data> *data_values,
    *      DDS_SampleInfoSeq *info_seq,
    *      const DDS_long max_samples,
    *      const DDS_ReadCondition a_condition);
    */
/* DDS_ReturnCode_t
    *     DDS_DataReader_take_w_condition
    *     (DDS_DataReader _this,
    *      DDS_sequence_<data> *data_values,

```

```

*         DDS_SampleInfoSeq *info_seq,
*         const DDS_long max_samples,
*         const DDS_ReadCondition a_condition);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_read_next_sample
*     (DDS_DataReader _this,
*      <data> *data_values,
*      DDS_SampleInfo *sample_info);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_take_next_sample
*     (DDS_DataReader _this,
*      <data> *data_values,
*      DDS_SampleInfo *sample_info);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_read_instance
*     (DDS_DataReader _this,
*      DDS_sequence_<data> *data_values,
*      DDS_SampleInfoSeq *info_seq,
*      const DDS_long max_samples,
*      const DDS_InstanceHandle_t a_handle,
*      const DDS_SampleStateMask sample_states,
*      const DDS_ViewStateMask view_states,
*      const DDS_InstanceStateMask instance_states);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_take_instance
*     (DDS_DataReader _this,
*      DDS_sequence_<data> *data_values,
*      DDS_SampleInfoSeq *info_seq,
*      const DDS_long max_samples,
*      const DDS_InstanceHandle_t a_handle,
*      const DDS_SampleStateMask sample_states,
*      const DDS_ViewStateMask view_states,
*      const DDS_InstanceStateMask instance_states);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_read_next_instance
*     (DDS_DataReader _this,
*      DDS_sequence_<data> *data_values,
*      DDS_SampleInfoSeq *info_seq,
*      const DDS_long max_samples,
*      const DDS_InstanceHandle_t a_handle,
*      const DDS_SampleStateMask sample_states,
*      const DDS_ViewStateMask view_states,
*      const DDS_InstanceStateMask instance_states);
*/
/* DDS_ReturnCode_t

```

```

*     DDS_DataReader_take_next_instance
*     (DDS_DataReader _this,
*       DDS_sequence_<data> *data_values,
*       DDS_SampleInfoSeq *info_seq,
*       const DDS_long max_samples,
*       const DDS_InstanceHandle_t a_handle,
*       const DDS_SampleStateMask sample_states,
*       const DDS_ViewStateMask view_states,
*       const DDS_InstanceStateMask instance_states);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_read_next_instance_w_condition
*     (DDS_DataReader _this,
*       DDS_sequence_<data> *data_values,
*       DDS_SampleInfoSeq *info_seq,
*       const DDS_long max_samples,
*       const DDS_InstanceHandle_t a_handle,
*       const DDS_ReadCondition a_condition);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_take_next_instance_w_condition
*     (DDS_DataReader _this,
*       DDS_sequence_<data> *data_values,
*       DDS_SampleInfoSeq *info_seq,
*       const DDS_long max_samples,
*       const DDS_InstanceHandle_t a_handle,
*       const DDS_ReadCondition a_condition);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_return_loan
*     (DDS_DataReader _this,
*       DDS_sequence_<data> *data_values,
*       DDS_SampleInfoSeq *info_seq);
*/
/* DDS_ReturnCode_t
*     DDS_DataReader_get_key_value
*     (DDS_DataReader _this,
*       <data> *key_holder,
*       const DDS_InstanceHandle_t handle);
*/
/* DDS_InstanceHandle_t
*     DDS_DataReader_lookup_instance
*/
*     (DDS_DataReader _this,
*       <data> *instance_data);
*/
* implemented API operations
*/
DDS_ReadCondition
DDS_DataReader_create_readcondition
(DDS_DataReader _this,

```



```

        const DDS_SampleStateMask sample_states,
        const DDS_ViewStateMask view_states,
        const DDS_InstanceStateMask instance_states);

DDS_QueryCondition
DDS_DataReader_create_querycondition
(DDS_DataReader _this,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states,
 const DDS_char *query_expression,
 const DDS_StringSeq *query_parameters);

DDS_ReturnCode_t
DDS_DataReader_delete_readcondition
(DDS_DataReader _this,
 const DDS_ReadCondition a_condition);

DDS_ReturnCode_t
DDS_DataReader_delete_contained_entities
(DDS_DataReader _this);

DDS_ReturnCode_t
DDS_DataReader_set_qos
(DDS_DataReader _this,
 const DDS_DataReaderQos *qos);

DDS_ReturnCode_t
DDS_DataReader_get_qos
(DDS_DataReader _this,
 DDS_DataReaderQos *qos);

DDS_ReturnCode_t
DDS_DataReader_set_listener
(DDS_DataReader _this,
 const struct DDS_DataReaderListener *a_listener,
 const DDS_StatusMask mask);

struct DDS_DataReaderListener
DDS_DataReader_get_listener
(DDS_DataReader _this);

DDS_TopicDescription
DDS_DataReader_get_topicdescription
(DDS_DataReader _this);

DDS_Subscriber
DDS_DataReader_get_subscriber
(DDS_DataReader _this);

```

```

DDS_ReturnCode_t
    DDS_DataReader_get_sample_rejected_status
        (DDS_DataReader _this,
         DDS_SampleRejectedStatus *status);

DDS_ReturnCode_t
    DDS_DataReader_get_liveliness_changed_status
        (DDS_DataReader _this,
         DDS_LivelinessChangedStatus *status);

DDS_ReturnCode_t
    DDS_DataReader_get_requested_deadline_missed_status
        (DDS_DataReader _this,
         DDS_RequestedDeadlineMissedStatus *status);

DDS_ReturnCode_t
    DDS_DataReader_get_requested_incompatible_qos_status
        (DDS_DataReader _this,
         DDS_RequestedIncompatibleQosStatus *status);

DDS_ReturnCode_t
    DDS_DataReader_get_subscription_matched_status
        (DDS_DataReader _this,
         DDS_SubscriptionMatchedStatus *status);

DDS_ReturnCode_t
    DDS_DataReader_get_sample_lost_status
        (DDS_DataReader _this,
         DDS_SampleLostStatus *status);

DDS_ReturnCode_t
    DDS_DataReader_wait_for_historical_data
        (DDS_DataReader _this,
         const DDS_Duration_t *max_wait);

DDS_ReturnCode_t
    DDS_DataReader_get_matched_publications
        (DDS_DataReader _this,
         DDS_InstanceHandleSeq *publication_handles);

DDS_ReturnCode_t
    DDS_DataReader_get_matched_publication_data
        (DDS_DataReader _this,
         DDS_PublicationBuiltinTopicData *publication_data,
         const DDS_InstanceHandle_t publication_handle);

```

The next paragraphs describe the usage of all `DDS_DataReader` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited. The abstract operations are listed but not fully

described because they are not implemented in this specific class. The full description of these operations is located in the subclasses that contain the data type specific implementation of these operations.

3.5.2.2 DDS_DataReader_create_querycondition

Synopsis

```
#include <dds_dcps.h>
DDS_QueryCondition
DDS_DataReader_create_querycondition
(DDS_DataReader _this,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states,
 const DDS_char *query_expression,
 const DDS_StringSeq *query_parameters);
```

Description

This operation creates a new DDS_QueryCondition for the DDS_DataReader.

Parameters

- in DDS_DataReader _this* - the DDS_DataReader object on which the operation is operated.
- in const DDS_SampleStateMask sample_states* - a mask, which selects only those samples with the desired sample states.
- in const DDS_ViewStateMask view_states* - a mask, which selects only those samples with the desired view states.
- in const DDS_InstanceStateMask instance_states* - a mask, which selects only those samples with the desired instance states.
- in const DDS_char *query_expression* - the query string, which must be a subset of the SQL query language as specified in Appendix H, *DCPS Queries and Filters*.
- in const DDS_StringSeq *query_parameters* - a sequence of strings which are the parameters used in the SQL query string (i.e., the “%n” tokens in the expression). The number of values in *query_parameters* must be equal or greater than the highest referenced %n token in the *query_expression* (e.g. if %1 and %8 are used as parameter in the *query_expression*, the *query_parameters* should at least contain $n+1 = 9$ values).

Return Value

DDS_QueryCondition - Result value is a pointer to the DDS_QueryCondition. When the operation fails, the DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation creates a new `DDS_QueryCondition` for the `DDS_DataReader`. The returned `DDS_QueryCondition` is attached (and belongs) to the `DDS_DataReader`. When the operation fails, the `DDS_OBJECT_NIL` pointer is returned. To delete the `DDS_QueryCondition` the operation `DDS_DataReader_delete_readcondition` or `DDS_DataReader_delete_contained_entities` must be used.

State Masks

The result of the `DDS_QueryCondition` also depends on the selection of samples determined by three masks:

- `sample_states` is the mask, which selects only those samples with the desired sample states `DDS_READ_SAMPLE_STATE`, `DDS_NOT_READ_SAMPLE_STATE` or both
- `view_states` is the mask, which selects only those samples with the desired view states `DDS_NEW_VIEW_STATE`, `DDS_NOT_NEW_VIEW_STATE` or both
- `instance_states` is the mask, which selects only those samples with the desired instance states `DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or a combination of these.

SQL expression

The SQL query string is set by `query_expression` which must be a subset of the SQL query language. In this query expression, parameters may be used, which must be set in the sequence of strings defined by the parameter `query_parameters`. A parameter is a string which can define an integer, float, string or enumeration. The number of values in `query_parameters` must be equal or greater than the highest referenced `%n` token in the `query_expression` (e.g. if `%1` and `%8` are used as parameter in the `query_expression`, the `query_parameters` should at least contain `n+1 = 9` values).

3.5.2.3 `DDS_DataReader_create_readcondition`

Synopsis

```
#include <dds_dcps.h>
DDS_ReadCondition
    DDS_DataReader_create_readcondition
        (DDS_DataReader _this,
         const DDS_SampleStateMask sample_states,
         const DDS_ViewStateMask view_states,
         const DDS_InstanceStateMask instance_states);
```

Description

This operation creates a new `DDS_ReadCondition` for the `DDS_DataReader`.

Parameters

in `DDS_DataReader _this` - the `DDS_DataReader` object on which the operation is operated.

in `const DDS_SampleStateMask sample_states` - a mask, which selects only those samples with the desired sample states.

in `const DDS_ViewStateMask view_states` - a mask, which selects only those samples with the desired view states.

in `const DDS_InstanceStateMask instance_states` - a mask, which selects only those samples with the desired instance states.

Return Value

`DDS_ReadCondition` - Result value is a pointer to the `DDS_ReadCondition`.
When the operation fails, the `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_ReadCondition` for the `DDS_DataReader`. The returned `DDS_ReadCondition` is attached (and belongs) to the `DDS_DataReader`. When the operation fails, the `DDS_OBJECT_NIL` pointer is returned. To delete the `DDS_ReadCondition` the operation `DDS_DataReader_delete_readcondition` or `DDS_DataReader_delete_contained_entities` must be used.

State Masks

The result of the `DDS_ReadCondition` depends on the selection of samples determined by three masks:

- `sample_states` is the mask, which selects only those samples with the desired sample states `DDS_READ_SAMPLE_STATE`, `DDS_NOT_READ_SAMPLE_STATE` or both
- `view_states` is the mask, which selects only those samples with the desired view states `DDS_NEW_VIEW_STATE`, `DDS_NOT_NEW_VIEW_STATE` or both
- `instance_states` is the mask, which selects only those samples with the desired instance states `DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or a combination of these.

3.5.2.4 DDS_DataReader_delete_contained_entities

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_delete_contained_entities
        (DDS_DataReader _this);
```

Description

This operation deletes all the DDS_Entity objects that were created by means of one of the “create_” operations on the DDS_DataReader.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_PRECONDITION_NOT_MET.

Detailed Description

This operation deletes all the DDS_Entity objects that were created by means of one of the “create_” operations on the DDS_DataReader. In other words, it deletes all DDS_QueryCondition and DDS_ReadCondition objects contained by the DDS_DataReader.



NOTE: The operation will return DDS_PRECONDITION_NOT_MET if the any of the contained entities is in a state where it cannot be deleted. In such cases, the operation does not roll back any entity deletions performed prior to the detection of the problem.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the contained DDS_Entity objects are deleted and the application may delete the DDS_DataReader.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.

- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one or more of the contained entities are in a state where they cannot be deleted.

3.5.2.5 DDS_DataReader_delete_readcondition

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_delete_readcondition
        (DDS_DataReader _this,
         const DDS_ReadCondition a_condition);
```

Description

This operation deletes a *DDS_ReadCondition* or *DDS_QueryCondition* which is attached to the *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

in const DDS_ReadCondition a_condition - a pointer to the *DDS_ReadCondition* or *DDS_QueryCondition* which is to be deleted.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_PRECONDITION_NOT_MET*.

Detailed Description

This operation deletes a *DDS_ReadCondition* or *DDS_QueryCondition* which is attached to the *DDS_DataReader*. Since a *DDS_QueryCondition* is a specialized *DDS_ReadCondition*, the operation can also be used to delete a *DDS_QueryCondition*. A *DDS_ReadCondition* or *DDS_QueryCondition* cannot be deleted when it is not attached to this *DDS_DataReader*. When the operation is called on a *DDS_ReadCondition* or *DDS_QueryCondition* which was not attached to this *DDS_DataReader*, the operation returns *DDS_RETCODE_PRECONDITION_NOT_MET*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the *DDS_ReadCondition* or *DDS_QueryCondition* is deleted.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - the parameter *a_condition* is not a valid *DDS_ReadCondition*.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - the operation is called on a different *DDS_DataReader*, as used when the *DDS_ReadCondition* or *DDS_QueryCondition* was created.

3.5.2.6 *DDS_DataReader_enable* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_enable
        (DDS_DataReader _this);
```

3.5.2.7 *DDS_DataReader_get_key_value* (abstract)

This abstract operation is defined as a generic operation, which is implemented by the *<NameSpace>_<type>DataReader* class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type *Foo* (defined in the module *SPACE*) derived *SPACE_FooDataReader* class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_key_value
        (DDS_DataReader _this,
         <data> *key_holder,
         const DDS_InstanceHandle_t handle);
```


3.5.2.8 DDS_DataReader_get_listener

Synopsis

```
#include <dds_dcps.h>
struct DDS_DataReaderListener
    DDS_DataReader_get_listener
        (DDS_DataReader _this);
```

Description

This operation allows access to a DDS_DataReaderListener.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

Return Value

struct DDS_DataReaderListener - result is a pointer to the DDS_DataReaderListener attached to the DDS_DataReader.

Detailed Description

This operation allows access to a DDS_DataReaderListener attached to the DDS_DataReader. When no DDS_DataReaderListener was attached to the DDS_DataReader, the DDS_OBJECT_NIL pointer is returned.

3.5.2.9 DDS_DataReader_get_liveliness_changed_status

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_liveliness_changed_status
        (DDS_DataReader _this,
         DDS_LivelinessChangedStatus *status);
```

Description

This operation obtains the DDS_LivelinessChangedStatus struct of the DDS_DataReader.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

*inout DDS_LivelinessChangedStatus *status* - the contents of the DDS_LivelinessChangedStatus struct of the DDS_DataReader will be copied into the location specified by status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation obtains the *DDS_LivelinessChangedStatus* struct of the *DDS_DataReader*. This struct contains the information whether the liveliness of one or more *DDS_DataWriter* objects that were writing instances read by the *DDS_DataReader* has changed. In other words, some *DDS_DataWriter* have become “alive” or “not alive”.

The *DDS_LivelinessChangedStatus* can also be monitored using a *DDS_DataReaderListener* or by using the associated *DDS_StatusCondition*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the current *DDS_LivelinessChangedStatus* of this *DDS_DataReader* has successfully been copied into the specified status parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.10 *DDS_DataReader_get_matched_publication_data*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_matched_publication_data
        (DDS_DataReader _this,
         DDS_PublicationBuiltinTopicData *publication_data,
         const DDS_InstanceHandle_t publication_handle);
```

Description

This operation retrieves information on the specified publication that is currently “associated” with the *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

*inout DDS_PublicationBuiltinTopicData *publication_data* - a pointer to the sample in which the information about the specified publication is to be stored.

in const DDS_InstanceHandle_t publication_handle - a handle to the publication whose information needs to be retrieved.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES or DDS_RETCODE_NOT_ENABLED.

Detailed Description

This operation retrieves information on the specified publication that is currently “associated” with the DDS_DataReader. That is, a publication with a matching Topic and compatible QoS that the application has not indicated should be “ignored” by means of the DDS_DomainParticipant_ignore_publication operation.

The *publication_handle* must correspond to a publication currently associated with the DDS_DataReader, otherwise the operation will fail and return DDS_RETCODE_BAD_PARAMETER. The operation *DDS_DataReader_get_matched_publications* can be used to find the publications that are currently matched with the DDS_DataReader.

The operation may also fail if the infrastructure does not hold the information necessary to fill in the *publication_data*. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the *NetworkingService/Discovery/enabled* property in the Deployment Manual for more information about this subject.) In this case the operation will return DDS_RETCODE_UNSUPPORTED.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the information on the specified publication has successfully been retrieved.
- *DDS_RETCODE_ERROR* - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_UNSUPPORTED* - OpenSplice is configured not to maintain the information about “associated” publications.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *DDS_DataReader* is not enabled.

3.5.2.11 DDS_DataReader_get_matched_publications

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_matched_publications
        (DDS_DataReader _this,
         DDS_InstanceHandleSeq *publication_handles);
```

Description

This operation retrieves the list of publications currently "associated" with the *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

*inout DDS_InstanceHandleSeq *publication_handles* - a sequence which is used to pass the list of all associated publications.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES* or *DDS_RETCODE_NOT_ENABLED*.

Detailed Description

This operation retrieves the list of publications currently "associated" with the *DDS_DataReader*. That is, subscriptions that have a matching Topic and compatible QoS that the application has not indicated should be “ignored” by means of the *DDS_DomainParticipant_ignore_publication* operation.

The `publication_handles` sequence and its buffer may be pre-allocated by the application and therefore must either be re-used in a subsequent invocation of the `DDS_DataReader_get_matched_publications` operation or be released by calling `DDS_free` on the returned `publication_handles`. If the pre-allocated sequence is not big enough to hold the number of associated publications, the sequence will automatically be (re-)allocated to fit the required size.

The handles returned in the `publication_handles` sequence are the ones that are used by the DDS implementation to locally identify the corresponding matched DataWriter entities. You can access more detailed information about a particular publication by passing its `publication_handle` to either the `DDS_DataReader_get_matched_publication_data` operation or to the `DDS_PublicationBuiltinTopicDataDataReader_read_instance` operation on the built-in reader for the “DCPSPublication” topic.



Be aware that since `DDS_InstanceHandle_t` is an opaque datatype, it does not necessarily mean that the handles obtained from the `DDS_DataReader_get_matched_publications` operation have the same value as the ones that appear in the `instance_handle` field of the `DDS_SampleInfo` when retrieving the publication info through corresponding “DCPSPublication” built-in reader. You can’t just compare two handles to determine whether they represent the same publication. If you want to know whether two handles actually do represent the same publication, use both handles to retrieve their corresponding `DDS_PublicationBuiltinTopicData` samples and then compare the key field of both samples.

The operation may fail if the infrastructure does not locally maintain the connectivity information. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the list of associated publications has successfully been obtained.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_UNSUPPORTED` - OpenSplice is configured not to maintain the information about “associated” publications.

- *DDS_RETCODE_ALREADY_DELETED* - the DDS_DataReader has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the DDS_DataReader is not enabled.

3.5.2.12 DDS_DataReader_get_qos

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_qos
        (DDS_DataReader _this,
         DDS_DataReaderQos *qos);
```

Description

This operation allows access to the existing set of QoS policies for a DDS_DataReader.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

*inout DDS_DataReaderQos *qos* - a pointer to the destination DDS_DataReaderQos struct in which the QoSPolicy settings will be copied.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation allows access to the existing set of QoS policies of a DDS_DataReader on which this operation is used. This DDS_DataReaderQos is stored at the location pointed to by the qos parameter.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the existing set of QoS policy values applied to this DDS_DataReader has successfully been copied into the specified DDS_DataReaderQos parameter.

- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.13 *DDS_DataReader_get_requested_deadline_missed_status*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_requested_deadline_missed_status
        (DDS_DataReader _this,
         DDS_RequestedDeadlineMissedStatus *status);
```

Description

This operation obtains the *DDS_RequestedDeadlineMissedStatus* struct of the *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

*inout DDS_RequestedDeadlineMissedStatus *status* - the contents of the *DDS_RequestedDeadlineMissedStatus* struct of the *DDS_DataReader* will be copied into the location specified by *status*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation obtains the *DDS_RequestedDeadlineMissedStatus* struct of the *DDS_DataReader*. This struct contains the information whether the deadline that the *DDS_DataReader* was expecting through its *DDS_DeadlineQosPolicy* was not respected for a specific instance.

The *DDS_RequestedDeadlineMissedStatus* can also be monitored using a *DDS_DataReaderListener* or by using the associated *DDS_StatusCondition*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the current *DDS_RequestedDeadlineMissedStatus* of this *DDS_DataReader* has successfully been copied into the specified status parameter.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.14 *DDS_DataReader_get_requested_incompatible_qos_status*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_requested_incompatible_qos_status
        (DDS_DataReader _this,
         DDS_RequestedIncompatibleQosStatus *status);
```

Description

This operation obtains the *DDS_RequestedIncompatibleQosStatus* struct of the *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

*inout DDS_RequestedIncompatibleQosStatus *status* - the contents of the *DDS_RequestedIncompatibleQosStatus* struct of the *DDS_DataReader* will be copied into the location specified by *status*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation obtains the `DDS_RequestedIncompatibleQosStatus` struct of the `DDS_DataReader`. This struct contains the information whether a `QosPolicy` setting was incompatible with the offered `QosPolicy` setting.

The Request/Offering mechanism is applicable between the `DDS_DataWriter` and the `DDS_DataReader`. If the `QosPolicy` settings between `DDS_DataWriter` and `DDS_DataReader` are inconsistent, no communication between them is established. In addition the `DDS_DataWriter` will be informed via a `DDS_REQUESTED_INCOMPATIBLE_QOS` status change and the `DDS_DataReader` will be informed via an `DDS_OFFERED_INCOMPATIBLE_QOS` status change.

The `DDS_RequestedIncompatibleQosStatus` can also be monitored using a `DDS_DataReaderListener` or by using the associated `DDS_StatusCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the current `DDS_RequestedIncompatibleQosStatus` of this `DDS_DataReader` has successfully been copied into the specified status parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.15 `DDS_DataReader_get_sample_lost_status`

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_sample_lost_status
        (DDS_DataReader _this,
         DDS_SampleLostStatus *status);
```

Description

This operation obtains the `DDS_SampleLostStatus` struct of the `DDS_DataReader`.

Parameters

in `DDS_DataReader _this` - the `DDS_DataReader` object on which the operation is operated.

inout `DDS_SampleLostStatus *status` - the contents of the `DDS_SampleLostStatus` struct of the `DDS_DataReader` will be copied into the location specified by `status`.

NOTE: This operation is not yet implemented. It is scheduled for a future release. Until it is implemented all returned attribute values will be initialized to 0.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are: `DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation obtains the `DDS_SampleLostStatus` struct of the `DDS_DataReader`. This struct contains the information whether a sample have been lost. This only applies when the `DDS_ReliabilityQosPolicy` is set to `DDS_RELIABLE`. If the `DDS_ReliabilityQosPolicy` is set to `DDS_BEST_EFFORT` the Data Distribution Service will not report the loss of samples.

The `DDS_SampleLostStatus` can also be monitored using a `DDS_DataReaderListener` or by using the associated `DDS_StatusCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the current `DDS_SampleLostStatus` of this `DDS_DataReader` has successfully been copied into the specified status parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.16 DDS_DataReader_get_sample_rejected_status

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_sample_rejected_status
        (DDS_DataReader _this,
         DDS_SampleRejectedStatus *status);
```

Detailed Description

This operation obtains the DDS_SampleRejectedStatus struct of the DDS_DataReader.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

*inout DDS_SampleRejectedStatus *status* - the contents of the DDS_SampleRejectedStatus struct of the DDS_DataReader will be copied into the location specified by status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation obtains the DDS_SampleRejectedStatus struct of the DDS_DataReader. This struct contains the information whether a received sample has been rejected. Samples may be rejected by the DDS_DataReader when it runs out of resource_limits to store incoming samples. Usually this means that old samples need to be ‘consumed’ (for example by ‘taking’ them instead of ‘reading’ them) to make room for newly incoming samples.

The DDS_SampleRejectedStatus can also be monitored using a DDS_DataReaderListener or by using the associated DDS_StatusCondition.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the current DDS_SampleRejectedStatus of this DDS_DataReader has successfully been copied into the specified status parameter.

- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.17 *DDS_DataReader_get_status_changes* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusMask
    DDS_DataReader_get_status_changes
        (DDS_DataReader _this);
```

3.5.2.18 *DDS_DataReader_get_statuscondition* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_Entity* for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_StatusCondition
    DDS_DataReader_get_statuscondition
        (DDS_DataReader _this);
```

3.5.2.19 *DDS_DataReader_get_subscriber*

Synopsis

```
#include <dds_dcps.h>
DDS_Subscriber
    DDS_DataReader_get_subscriber
        (DDS_DataReader _this);
```

Description

This operation returns the *DDS_Subscriber* to which the *DDS_DataReader* belongs.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

Return Value

DDS_Subscriber - Return value is a pointer to the *DDS_Subscriber* to which the *DDS_DataReader* belongs.

Detailed Description

This operation returns the *DDS_Subscriber* to which the *DDS_DataReader* belongs, thus the *DDS_Subscriber* that has created the *DDS_DataReader*. If the *DDS_DataReader* is already deleted, the *DDS_OBJECT_NIL* pointer is returned.

3.5.2.20 DDS_DataReader_get_subscription_matched_status

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_get_subscription_matched_status
        (DDS_DataReader _this,
         DDS_SubscriptionMatchedStatus *status);
```

Description

This operation obtains the *DDS_SubscriptionMatchedStatus* struct of the *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

*inout DDS_SubscriptionMatchedStatus *status* - the contents of the *DDS_SubscriptionMatchedStatus* struct of the *DDS_DataReader* will be copied into the location specified by *status*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: *DDS_RETCODE_OK*, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED* or *DDS_RETCODE_OUT_OF_RESOURCES*.

Detailed Description

This operation obtains the *DDS_SubscriptionMatchedStatus* struct of the *DDS_DataReader*. This struct contains the information whether a new match has been discovered for the current subscription, or whether an existing match has ceased to exist.

This means that the status represents that either a `DataWriter` object has been discovered by the `DDS_DataReader` with the same Topic and a compatible Qos, or that a previously-discovered `DataWriter` has ceased to be matched to the current `DDS_DataReader`. A `DataWriter` may cease to match when it gets deleted, when it changes its Qos to a value that is incompatible with the current `DDS_DataReader` or when either the **`DDS_DataReader`** or the `DataWriter` has chosen to put its matching counterpart on its ignore-list using the `DDS_DomainParticipant_ignore_publication` or `DDS_DomainParticipant_ignore_subscription` operations.

The operation may fail if the infrastructure does not hold the information necessary to fill in the `DDS_SubscriptionMatchedStatus`. This is the case when `OpenSplice` is configured not to maintain discovery information in the `NetworkingService`. (See the description for the `NetworkingService/Discovery/enabled` property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

The `DDS_SubscriptionMatchedStatus` can also be monitored using a `DDS_DataReaderListener` or by using the associated `DDS_StatusCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the current `DDS_SubscriptionMatchedStatus` of this `DDS_DataReader` has successfully been copied into the specified status parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_UNSUPPORTED` - `OpenSplice` is configured not to maintain the information about “associated” publications.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.21 `DDS_DataReader_get_topicdescription`

Synopsis

```
#include <dds_dcps.h>
DDS_TopicDescription
    DDS_DataReader_get_topicdescription
```

```
(DDS_DataReader _this);
```

Description

This operation returns the `DDS_TopicDescription` which is associated with the `DDS_DataReader`.

Parameters

in `DDS_DataReader _this` - the `DDS_DataReader` object on which the operation is operated.

Return Value

`DDS_TopicDescription` - a pointer to the `DDS_TopicDescription` which is associated with the `DDS_DataReader`.

Detailed Description

This operation returns the `DDS_TopicDescription` which is associated with the `DDS_DataReader`, thus the `DDS_TopicDescription` with which the `DDS_DataReader` is created. If the `DDS_DataReader` is already deleted, the `DDS_OBJECT_NIL` pointer is returned.

3.5.2.22 DDS_DataReader_lookup_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
DDS_InstanceHandle_t
    DDS_DataReader_lookup_instance
        (DDS_DataReader _this,
         <data> *instance_data);
```

3.5.2.23 DDS_DataReader_read (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
```

```

DDS_ReturnCode_t
DDS_DataReader_read
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);

```

3.5.2.24 DDS_DataReader_read_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_read_instance
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);

```

3.5.2.25 DDS_DataReader_read_next_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_read_next_instance
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,

```



```
const DDS_SampleStateMask sample_states,
const DDS_ViewStateMask view_states,
const DDS_InstanceStateMask instance_states);
```

3.5.2.26 DDS_DataReader_read_next_instance_w_condition (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<Namespace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_read_next_instance_w_condition
        (DDS_DataReader _this,
         DDS_sequence_<data> *data_values,
         DDS_SampleInfoSeq *info_seq,
         const DDS_long max_samples,
         const DDS_InstanceHandle_t a_handle,
         const DDS_ReadCondition a_condition);
```

3.5.2.27 DDS_DataReader_read_next_sample (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<Namespace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_read_next_sample
        (DDS_DataReader _this,
         <data> *data_values,
         DDS_SampleInfo *sample_info);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.2.28 DDS_DataReader_read_w_condition (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<Namespace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_read_w_condition
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_ReadCondition a_condition);
```

3.5.2.29 DDS_DataReader_return_loan (abstract)

This abstract operation is defined as a generic operation, which is implemented by the <NameSpace>_<type>DataReader class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type Foo (defined in the module SPACE) derived SPACE_FooDataReader class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_return_loan
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq);
```

3.5.2.30 DDS_DataReader_set_listener

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_set_listener
(DDS_DataReader _this,
 const struct DDS_DataReaderListener *a_listener,
 const DDS_StatusMask mask);
```

Description

This operation attaches a DDS_DataReaderListener to the DDS_DataReader.

Parameters

in DDS_DataReader _this - the DDS_DataReader object on which the operation is operated.

*in const struct DDS_DataReaderListener *a_listener* - a pointer to the DDS_DataReaderListener instance, which will be attached to the DDS_DataReader.

in const DDS_StatusMask mask - a bit-mask in which each bit enables the invocation of the DDS_DataReaderListener for a certain status.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:

DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_UNSUPPORTED, DDS_RETCODE_ALREADY_DELETED or DDS_RETCODE_OUT_OF_RESOURCES.

Detailed Description

This operation attaches a DDS_DataReaderListener to the DDS_DataReader. Only one DDS_DataReaderListener can be attached to each DDS_DataReader. If a DDS_DataReaderListener was already attached, the operation will replace it with the new one. When *a_listener* is the DDS_OBJECT_NIL pointer, it represents a listener that is treated as a NOOP¹ for all statuses activated in the bitmask.

Communication Status

For each communication status, the StatusChangedFlag flag is initially set to FALSE. It becomes TRUE whenever that communication status changes. For each communication status activated in the *mask*, the associated DDS_DataReaderListener operation is invoked and the communication status is reset to FALSE, as the listener implicitly accesses the status which is passed as a parameter to that operation. The status is reset prior to calling the listener, so if the application calls the *get_<status_name>_status* from inside the listener it will see the status already reset. An exception to this rule is the DDS_OBJECT_NIL listener, which does not reset the communication statuses for which it is invoked.

The following statuses are applicable to the DDS_DataReaderListener:

DDS_REQUESTED_DEADLINE_MISSED_STATUS
 DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS
 DDS_SAMPLE_LOST_STATUS
 DDS_SAMPLE_REJECTED_STATUS
 DDS_DATA_AVAILABLE_STATUS
 DDS_LIVELINESS_CHANGED_STATUS
 DDS_SUBSCRIPTION_MATCHED_STATUS.



Be aware that the DDS_SUBSCRIPTION_MATCHED_STATUS is not applicable when the infrastructure does not have the information available to determine connectivity. This is the case when OpenSplice is configured not to maintain discovery information in the Networking Service. (See the description for the

1. Short for **No-Operation**, an instruction that performs nothing at all.

NetworkingService/Discovery/enabled property in the Deployment Manual for more information about this subject.) In this case the operation will return `DDS_RETCODE_UNSUPPORTED`.

Status bits are declared as a constant and can be used by the application in an OR operation to create a tailored mask. The special constant `DDS_STATUS_MASK_NONE` can be used to indicate that the created entity should not respond to any of its available statuses. The DDS will therefore attempt to propagate these statuses to its factory. The special constant `STATUS_MASK_ANY_V1_2` can be used to select all applicable statuses specified in the “Data Distribution Service for Real-time Systems Version 1.2” specification.

Status Propagation

In case a communication status is not activated in the mask of the `DDS_DataReaderListener`, the `DDS_SubscriberListener` of the containing `DDS_Subscriber` is invoked (if attached and activated for the status that occurred). This allows the application to set a default behaviour in the `DDS_SubscriberListener` of the containing `DDS_Subscriber` and a `DDS_DataReader` specific behaviour when needed. In case the communication status is not activated in the mask of the `DDS_SubscriberListener` as well, the communication status will be propagated to the `DDS_DomainParticipantListener` of the containing `DDS_DomainParticipant`. In case the `DDS_DomainParticipantListener` is also not attached or the communication status is not activated in its mask, the application is not notified of the change.

The statuses `DDS_DATA_ON_READERS_STATUS` and `DDS_DATA_AVAILABLE_STATUS` are “Read Communication Statuses” and are an exception to all other plain communication statuses: they have no corresponding status structure that can be obtained with a `get_<status_name>_status` operation and they are mutually exclusive. When new information becomes available to a `DataReader`, the Data Distribution Service will first look in an attached and activated `DDS_SubscriberListener` or `DDS_DomainParticipantListener` (in that order) for the `DDS_DATA_ON_READERS_STATUS`. In case the `DDS_DATA_ON_READERS_STATUS` can not be handled, the Data Distribution Service will look in an attached and activated `DDS_DataReaderListener`, `DDS_SubscriberListener` or `DDS_DomainParticipantListener` for the `DDS_DATA_AVAILABLE_STATUS` (in that order).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_DataReaderListener` is attached.
- `DDS_RETCODE_ERROR` - an internal error has occurred.

- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_UNSUPPORTED* - a status was selected that cannot be supported because the infrastructure does not maintain the required connectivity information.
- *DDS_RETCODE_ALREADY_DELETED* - the *DDS_DataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.

3.5.2.31 *DDS_DataReader_set_qos*

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_DataReader_set_qos
        (DDS_DataReader _this,
         const DDS_DataReaderQos *qos);
```

Description

This operation replaces the existing set of *QosPolicy* settings for a *DDS_DataReader*.

Parameters

in DDS_DataReader _this - the *DDS_DataReader* object on which the operation is operated.

*in const DDS_DataReaderQos *qos* - the new set of *QosPolicy* settings for the *DDS_DataReader*.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
DDS_RETCODE_OK, *DDS_RETCODE_ERROR*, *DDS_RETCODE_ILLEGAL_OPERATION*, *DDS_RETCODE_BAD_PARAMETER*, *DDS_RETCODE_UNSUPPORTED*, *DDS_RETCODE_ALREADY_DELETED*, *DDS_RETCODE_OUT_OF_RESOURCES*, *DDS_RETCODE_IMMUTABLE_POLICY* or *DDS_RETCODE_INCONSISTENT_POLICY*.

Detailed Description

This operation replaces the existing set of *QosPolicy* settings for a *DDS_DataReader*. The parameter *qos* contains the *QosPolicy* settings which is checked for self-consistency and mutability. When the application tries to change a

`QosPolicy` setting for an enabled `DDS_DataReader`, which can only be set before the `DDS_DataReader` is enabled, the operation will fail and a `DDS_RETCODE_IMMUTABLE_POLICY` is returned. In other words, the application must provide the presently set `QosPolicy` settings in case of the immutable `QosPolicy` settings. Only the mutable `QosPolicy` settings can be changed. When `qos` contains conflicting `QosPolicy` settings (not self-consistent), the operation will fail and a `DDS_RETCODE_INCONSISTENT_POLICY` is returned.

The set of `QosPolicy` settings specified by the `qos` parameter are applied on top of the existing QoS, replacing the values of any policies previously set (provided, the operation returned `DDS_RETCODE_OK`).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the new `DDS_DataReaderQos` is set.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the parameter `qos` is not a valid `DDS_DataReaderQos`. It contains a `QosPolicy` setting with an invalid `DDS_Duration_t` value, an enum value that is outside its legal boundaries or a sequence that has inconsistent memory settings.
- `DDS_RETCODE_UNSUPPORTED` - one or more of the selected `QosPolicy` values are currently not supported by OpenSplice.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_IMMUTABLE_POLICY` - the parameter `qos` contains an immutable `QosPolicy` setting with a different value than set during enabling of the `DDS_DataReader`.
- `DDS_RETCODE_INCONSISTENT_POLICY` - the parameter `qos` contains conflicting `QosPolicy` settings, e.g. a history depth that is higher than the specified resource limits.

3.5.2.32 DDS_DataReader_take (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
        DDS_ReturnCode_t
        DDS_DataReader_take
        (DDS_DataReader _this,
         DDS_sequence_<data> *data_values,
         DDS_SampleInfoSeq *info_seq,
         const DDS_long max_samples,
         const DDS_SampleStateMask sample_states,
         const DDS_ViewStateMask view_states,
         const DDS_InstanceStateMask instance_states);
```

3.5.2.33 DDS_DataReader_take_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
        DDS_ReturnCode_t
        DDS_DataReader_take_instance
        (DDS_DataReader _this,
         DDS_sequence_<data> *data_values,
         DDS_SampleInfoSeq *info_seq,
         const DDS_long max_samples,
         const DDS_InstanceHandle_t a_handle,
         const DDS_SampleStateMask sample_states,
         const DDS_ViewStateMask view_states,
         const DDS_InstanceStateMask instance_states);
```

3.5.2.34 DDS_DataReader_take_next_instance (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_take_next_instance
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);
```

3.5.2.35 DDS_DataReader_take_next_instance_w_condition (abstract)

This abstract operation is defined as a generic operation, which is implemented by the <NameSpace>_<type>DataReader class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type Foo (defined in the module SPACE) derived SPACE_FooDataReader class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_take_next_instance_w_condition
(DDS_DataReader _this,
 DDS_sequence_<data> *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_ReadCondition a_condition);
```

3.5.2.36 DDS_DataReader_take_next_sample (abstract)

This abstract operation is defined as a generic operation, which is implemented by the <NameSpace>_<type>DataReader class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type Foo (defined in the module SPACE) derived SPACE_FooDataReader class.

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
DDS_DataReader_take_next_sample
(DDS_DataReader _this,
 <data> *data_values,
 DDS_SampleInfo *sample_info);
```


NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.2.37 DDS_DataReader_take_w_condition (abstract)

This abstract operation is defined as a generic operation, which is implemented by the `<NameSpace>_<type>DataReader` class. Therefore, to use this operation, the data type specific implementation of this operation in its respective derived class must be used. For further explanation see the description for the fictional data type `Foo` (defined in the module `SPACE`) derived `SPACE_FooDataReader` class.

Synopsis

```
#include <dds_dcps.h>
        DDS_ReturnCode_t
        DDS_DataReader_take_w_condition
        (DDS_DataReader _this,
         DDS_sequence_<data> *data_values,
         DDS_SampleInfoSeq *info_seq,
         const DDS_long max_samples,
         const DDS_ReadCondition a_condition);
```

3.5.2.38 DDS_DataReader_wait_for_historical_data

Synopsis

```
#include <dds_dcps.h>
        DDS_ReturnCode_t
        DDS_DataReader_wait_for_historical_data
        (DDS_DataReader _this,
         const DDS_Duration_t *max_wait);
```

Description

This operation will block the application thread until all “historical” data is received.

Parameters

in `DDS_DataReader _this` - the `DDS_DataReader` object on which the operation is operated.

in `const DDS_Duration_t *max_wait` - the maximum duration to block for the `DDS_DataReader_wait_for_historical_data`, after which the application thread is unblocked. The special constant `DDS_DURATION_INFINITE` can be used when the maximum waiting time does not need to be bounded.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED or DDS_RETCODE_TIMEOUT.

Detailed Description

This operation behaves differently for *DDS_DataReader* objects which have a non-*DDS_VOLATILE_DURABILITY_QOS* *DDS_DurabilityQosPolicy* and for *DDS_DataReader* objects which have a *DDS_VOLATILE_DURABILITY_QOS* *DDS_DurabilityQosPolicy*.

As soon as an application enables a non-*DDS_VOLATILE_DURABILITY_QOS* *DDS_DataReader* it will start receiving both “historical” data, i.e. the data that was written prior to the time the *DDS_DataReader* joined the domain, as well as any new data written by the *DDS_DataWriter* objects. There are situations where the application logic may require the application to wait until all “historical” data is received. This is the purpose of the *DDS_DataReader_wait_for_historical_data* operation.

As soon as an application enables a *DDS_VOLATILE_DURABILITY_QOS* *DataReader* it will not start receiving “historical” data but only new data written by the *DDS_DataWriter* objects. By calling *DDS_DataReader_wait_for_historical_data* the *DDS_DataReader* explicitly requests the Data Distribution Service to start receiving also the “historical” data and to wait until either all “historical” data is received, or the duration specified by the *max_wait* parameter has elapsed, whichever happens first.

Thread blocking

The operation *DDS_DataReader_wait_for_historical_data* blocks the calling thread until either all “historical” data is received, or the duration specified by the *max_wait* parameter elapses, whichever happens first. A return value of *DDS_RETCODE_OK* indicates that all the “historical” data was received; a return value of *DDS_RETCODE_TIMEOUT* indicates that *max_wait* elapsed before all the data was received.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - the “historical” data is received.
- *DDS_RETCODE_ERROR* - an internal error has occurred.

- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_DataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `DDS_DataReader` is not enabled.
- `DDS_RETCODE_TIMEOUT` - not all data is received before `max_wait` elapsed.

3.5.2.39 Class `SPACE_FooDataReader`

The pre-processor generates from IDL type descriptions the application `<NameSpace>_<type>DataReader` classes. For each application data type that is used as `DDS_Topic` data type, a typed class `<NameSpace>_<type>DataReader` is derived from the `DDS_DataReader` class. In this paragraph, the class `SPACE_FooDataReader` describes the operations of these derived `<NameSpace>_<type>DataReader` classes as an example for the fictional application type `Foo` (defined in the module `SPACE`).

For instance, for an application, the definitions are located in the `Space.idl` file. The pre-processor will generate a `Space.h` include file.

State masks

A `SPACE_FooDataReader` refers to exactly one `DDS_TopicDescription` (either a `DDS_Topic`, a `DDS_ContentFilteredTopic` or a `DDS_MultiTopic`) that identifies the data to be read. Therefore it refers to exactly one data type. The `DDS_Topic` must exist prior to the `SPACE_FooDataReader` creation. The `SPACE_FooDataReader` may give access to several instances of the data type, which are distinguished from each other by their key. The `SPACE_FooDataReader` is attached to exactly one `DDS_Subscriber` which acts as a factory for it.

The interface description of this class is as follows:

```
/*
 * interface SPACE_FooDataReader
 */
/*
 * inherited from class DDS_Entity
 */
/* DDS_StatusCondition
 *     SPACE_FooDataReader_get_statuscondition
 *     (SPACE_FooDataReader _this);
 */
/* DDS_StatusMask
 *     SPACE_FooDataReader_get_status_changes
```

```

*      (SPACE_FooDataReader _this);
*/
/* DDS_ReturnCode_t
*      SPACE_FooDataReader_enable
*      (SPACE_FooDataReader _this);
*/
/*
* inherited from class DDS_DataReader
*/
/* DDS_ReadCondition
*      SPACE_FooDataReader_create_readcondition
*      (SPACE_FooDataReader _this,
*       const DDS_SampleStateMask sample_states,
*       const DDS_ViewStateMask view_states,
*       const DDS_InstanceStateMask instance_states);
*/

/* DDS_QueryCondition
*      SPACE_FooDataReader_create_querycondition
*      (SPACE_FooDataReader _this,
*       const DDS_SampleStateMask sample_states,
*       const DDS_ViewStateMask view_states,
*       const DDS_InstanceStateMask instance_states,
*       const DDS_char *query_expression,
*       const DDS_StringSeq *query_parameters);
*/

/* DDS_ReturnCode_t
*      SPACE_FooDataReader_delete_readcondition
*      (SPACE_FooDataReader _this,
*       const DDS_ReadCondition a_condition);
*/

/* DDS_ReturnCode_t
*      SPACE_FooDataReader_delete_contained_entities
*      (SPACE_FooDataReader _this);
*/

/* DDS_ReturnCode_t
*      SPACE_FooDataReader_set_qos
*      (SPACE_FooDataReader _this,
*       const DDS_DataReaderQos *qos);
*/

/* DDS_ReturnCode_t
*      SPACE_FooDataReader_get_qos
*      (SPACE_FooDataReader _this,
*       SPACE_FooDataReaderQos *qos);
*/

```

```

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_set_listener
 *   (SPACE_FooDataReader _this,
 *    const struct DDS_DataReaderListener *a_listener,
 *    const DDS_StatusMask mask);
 */

/* struct SPACE_FooDataReaderListener
 *   SPACE_FooDataReader_get_listener
 *   (SPACE_FooDataReader _this);
 */

/* DDS_TopicDescription
 *   SPACE_FooDataReader_get_topicdescription
 *   (SPACE_FooDataReader _this);
 */

/* DDS_Subscriber
 *   SPACE_FooDataReader_get_subscriber
 *   (SPACE_FooDataReader _this);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_sample_rejected_status
 *   (SPACE_FooDataReader _this,
 *    DDS_SampleRejectedStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_liveliness_changed_status
 *   (SPACE_FooDataReader _this,
 *    DDS_LivelinessChangedStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_requested_deadline_missed_status
 *   (SPACE_FooDataReader _this,
 *    DDS_RequestedDeadlineMissedStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_requested_incompatible_qos_status
 *   (SPACE_FooDataReader _this,
 *    DDS_RequestedIncompatibleQosStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_subscription_matched_status
 *   (SPACE_FooDataReader _this,
 *    DDS_SubscriptionMatchedStatus *status);

```

```

*/

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_sample_lost_status
 *   (SPACE_FooDataReader _this,
 *    DDS_SampleLostStatus *status);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_wait_for_historical_data
 *   (SPACE_FooDataReader _this,
 *    const DDS_Duration_t *max_wait);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_matched_publications
 *   (SPACE_FooDataReader _this,
 *    DDS_InstanceHandleSeq *publication_handles);
 */

/* DDS_ReturnCode_t
 *   SPACE_FooDataReader_get_matched_publication_data
 *   (SPACE_FooDataReader _this,
 *    DDS_PublicationBuiltinTopicData *publication_data,
 *    const DDS_InstanceHandle_t publication_handle);
 */
/*
 * implemented API operations
 */
    DDS_ReturnCode_t
        SPACE_FooDataReader_read
        (SPACE_FooDataReader _this,
         DDS_sequence_Foo *data_values,
         DDS_SampleInfoSeq *info_seq,
         const DDS_long max_samples,
         const DDS_SampleStateMask sample_states,
         const DDS_ViewStateMask view_states,
         const DDS_InstanceStateMask instance_states);
    DDS_ReturnCode_t
        SPACE_FooDataReader_take
        (SPACE_FooDataReader _this,
         DDS_sequence_Foo *data_values,
         DDS_SampleInfoSeq *info_seq,
         const DDS_long max_samples,
         const DDS_SampleStateMask sample_states,
         const DDS_ViewStateMask view_states,
         const DDS_InstanceStateMask instance_states);
    DDS_ReturnCode_t
        SPACE_FooDataReader_read_w_condition
        (SPACE_FooDataReader _this,

```

```

        DDS_sequence_Foo *data_values,
        DDS_SampleInfoSeq *info_seq,
        const DDS_long max_samples,
        const DDS_ReadCondition a_condition);
DDS_ReturnCode_t
SPACE_FooDataReader_take_w_condition
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_ReadCondition a_condition);
DDS_ReturnCode_t
SPACE_FooDataReader_read_next_sample
(SPACE_FooDataReader _this,
 Foo *data_values,
 DDS_SampleInfo *sample_info);
DDS_ReturnCode_t
SPACE_FooDataReader_take_next_sample
(SPACE_FooDataReader _this,
 Foo *data_values,
 DDS_SampleInfo *sample_info);
DDS_ReturnCode_t
SPACE_FooDataReader_read_instance
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);
DDS_ReturnCode_t
SPACE_FooDataReader_take_instance
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);
DDS_ReturnCode_t
SPACE_FooDataReader_read_next_instance
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,

```

```

        const DDS_InstanceStateMask instance_states);
DDS_ReturnCode_t
SPACE_FooDataReader_take_next_instance
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);
DDS_ReturnCode_t
SPACE_FooDataReader_read_next_instance_w_condition
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_ReadCondition a_condition);
DDS_ReturnCode_t
SPACE_FooDataReader_take_next_instance_w_condition
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_ReadCondition a_condition);
DDS_ReturnCode_t
SPACE_FooDataReader_return_loan
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq);
DDS_ReturnCode_t
SPACE_FooDataReader_get_key_value
(SPACE_FooDataReader _this,
 Foo *key_holder,
 const DDS_InstanceHandle_t handle);
DDS_InstanceHandle_t
SPACE_FooDataReader_lookup_instance
(SPACE_FooDataReader _this,
 Foo *instance_data);

```

The next paragraphs describe the usage of all `SPACE_FooDataReader` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.5.2.40 SPACE_FooDataReader_create_querycondition (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_QueryCondition
SPACE_FooDataReader_create_querycondition
(SPACE_FooDataReader _this,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states,
 const DDS_char *query_expression,
 const DDS_StringSeq *query_parameters);
```

3.5.2.41 SPACE_FooDataReader_create_readcondition (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReadCondition
SPACE_FooDataReader_create_readcondition
(SPACE_FooDataReader _this,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);
```

3.5.2.42 SPACE_FooDataReader_delete_contained_entities (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_delete_contained_entities
(SPACE_FooDataReader _this);
```

3.5.2.43 SPACE_FooDataReader_delete_readcondition (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
```

```
SPACE_FooDataReader_delete_readcondition
(SPACE_FooDataReader _this,
 const DDS_ReadCondition a_condition);
```

3.5.2.44 SPACE_FooDataReader_enable (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Entity` for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_enable
(SPACE_FooDataReader _this);
```

3.5.2.45 SPACE_FooDataReader_get_key_value

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_get_key_value
(SPACE_FooDataReader _this,
 Foo *key_holder,
 const DDS_InstanceHandle_t handle);
```

Description

This operation retrieves the key value of a specific instance.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

inout `Foo *key_holder` - the sample in which the key values are stored.

in `const DDS_InstanceHandle_t handle` - the handle to the instance from which to get the key value.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation retrieves the key value of the instance pointed to by `instance_handle`. When the operation is called with an `DDS_HANDLE NIL` handle value as an `instance_handle`, the operation will return `DDS_RETCODE_BAD_PARAMETER`. The operation will only fill the fields that form the key inside the `key_holder` instance. This means that the non-key fields are not applicable and may contain garbage.

The operation must only be called on registered instances. Otherwise the operation returns the error `DDS_RETCODE_PRECONDITION_NOT_MET`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `key_holder` instance contains the key values of the instance.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - handle is not a valid handle or `key_holder` is not a valid pointer.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataReader` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - this instance is not registered.

3.5.2.46 `SPACE_FooDataReader_get_listener` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataReader` for further explanation.

Synopsis

```
#include <Space.h>
struct SPACE_FooDataReaderListener
    SPACE_FooDataReader_get_listener
        (SPACE_FooDataReader _this);
```

3.5.2.47 `SPACE_FooDataReader_get_liveliness_changed_status` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataReader` for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataReader_get_liveliness_changed_status
        (SPACE_FooDataReader _this,
         DDS_LivelinessChangedStatus *status);
```

3.5.2.48 SPACE_FooDataReader_get_matched_publication_data (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataReader_get_matched_publication_data
        (SPACE_FooDataReader _this,
         DDS_PublicationBuiltinTopicData *publication_data,
         const DDS_InstanceHandle_t publication_handle);
```

3.5.2.49 SPACE_FooDataReader_get_matched_publications (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataReader_get_matched_publications
        (SPACE_FooDataReader _this,
         DDS_InstanceHandleSeq *publication_handles);
```

3.5.2.50 SPACE_FooDataReader_get_qos (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
    SPACE_FooDataReader_get_qos
        (SPACE_FooDataReader _this,
         DDS_DataReaderQos *qos);
```

3.5.2.51 SPACE_FooDataReader_get_requested_deadline_missed_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_get_requested_deadline_missed_status
(SPACE_FooDataReader _this,
 DDS_RequestedDeadlineMissedStatus *status);
```

3.5.2.52 SPACE_FooDataReader_get_requested_incompatible_qos_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_get_requested_incompatible_qos_status
(SPACE_FooDataReader _this,
 DDS_RequestedIncompatibleQosStatus *status);
```

3.5.2.53 SPACE_FooDataReader_get_sample_lost_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_get_sample_lost_status
(SPACE_FooDataReader _this,
 DDS_SampleLostStatus *status);
```

3.5.2.54 SPACE_FooDataReader_get_sample_rejected_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_get_sample_rejected_status
(SPACE_FooDataReader _this,
 DDS_SampleRejectedStatus *status);
```

3.5.2.55 SPACE_FooDataReader_get_status_changes (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```
#include <Space.h>
DDS_StatusMask
SPACE_FooDataReader_get_status_changes
    (SPACE_FooDataReader _this);
```

3.5.2.56 SPACE_FooDataReader_get_statuscondition (inherited)

This operation is inherited and therefore not described here. See the class DDS_Entity for further explanation.

Synopsis

```
#include <Space.h>
DDS_StatusCondition
SPACE_FooDataReader_get_statuscondition
    (SPACE_FooDataReader _this);
```

3.5.2.57 SPACE_FooDataReader_get_subscriber (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_Subscriber
SPACE_FooDataReader_get_subscriber
    (SPACE_FooDataReader _this);
```

3.5.2.58 SPACE_FooDataReader_get_subscription_matched_status (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_get_subscription_matched_status
    (SPACE_FooDataReader _this,
     DDS_SubscriptionMatchedStatus *status);
```

3.5.2.59 SPACE_FooDataReader_get_topicdescription (inherited)

This operation is inherited and therefore not described here. See the class DDS_DataReader for further explanation.

Synopsis

```
#include <Space.h>
DDS_TopicDescription
```

```
SPACE_FooDataReader_get_topicdescription
(SPACE_FooDataReader _this);
```

3.5.2.60 SPACE_FooDataReader_lookup_instance

Synopsis

```
#include <Space.h>
DDS_InstanceHandle_t
SPACE_FooDataReader_lookup_instance
(SPACE_FooDataReader _this,
 Foo *instance_data);
```

Description

This operation returns the value of the instance handle which corresponds to the `instance_data`.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

in `Foo *instance_data` - the instance for which the corresponding instance handle needs to be looked up.

Return Value

`DDS_InstanceHandle_t` - Result value is the instance handle which corresponds to the `instance_data`.

Detailed Description

This operation returns the value of the instance handle which corresponds to the `instance_data`. The instance handle can be used in read operations that operate on a specific instance. Note that `DDS_DataReader` instance handles are local, and are not interchangeable with `DDS_DataWriter` instance handles nor with instance handles of an other `DDS_DataReader`. If the `DDS_DataReader` is already deleted, the handle value `DDS_HANDLE_NIL` is returned.

3.5.2.61 SPACE_FooDataReader_read

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_read
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
```

```
const DDS_SampleStateMask sample_states,
const DDS_ViewStateMask view_states,
const DDS_InstanceStateMask instance_states);
```

Description

This operation reads a sequence of `Foo` samples from the `SPACE_FooDataReader`.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

inout `DDS_sequence_Foo *data_values` - the returned sample data sequence. `data_values` is also used as an input to control the behaviour of this operation.

inout `DDS_SampleInfoSeq *info_seq` - the returned `DDS_SampleInfo` structure sequence. `info_seq` is also used as an input to control the behaviour of this operation.

in `const DDS_long max_samples` - the maximum number of samples that is returned.

in `const DDS_SampleStateMask sample_states` - a mask, which selects only those samples with the desired sample states.

in `const DDS_ViewStateMask view_states` - a mask, which selects only those samples with the desired view states.

in `const DDS_InstanceStateMask instance_states` - a mask, which selects only those samples with the desired instance states.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED`, `DDS_RETCODE_PRECONDITION_NOT_MET` or `DDS_RETCODE_NO_DATA`.

Detailed Description

This operation reads a sequence of `Foo` samples from the `SPACE_FooDataReader`. The data is returned by the parameters `data_values` and `info_seq`. The number of samples that is returned is limited by the parameter `max_samples`. This operation is part of the specialized class which is generated for the particular

application data type (in this case type `Foo`) that is being read. If the `SPACE_FooDataReader` has no samples that meet the constraints, the return value is `DDS_RETCODE_NO_DATA`.

State masks

The `SPACE_FooDataReader_read` operation depends on a selection of the samples by using three masks:

- `sample_states` is the mask, which selects only those samples with the desired sample states `DDS_READ_SAMPLE_STATE`, `DDS_NOT_READ_SAMPLE_STATE` or both
- `view_states` is the mask, which selects only those samples with the desired view states `DDS_NEW_VIEW_STATE`, `DDS_NOT_NEW_VIEW_STATE` or both
- `instance_states` is the mask, which selects only those samples with the desired instance states `DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or a combination of these.

Destination Order

In any case, the relative order between the samples of one instance is consistent with the `DDS_DestinationOrderQosPolicy` of the `DDS_Subscriber`.

When the `DDS_DestinationOrderQosPolicy` kind is `DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS`, the samples belonging to the same instances will appear in the relative order in which they were received (FIFO)

When the `DDS_DestinationOrderQosPolicy` kind is `DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS`, the samples belonging to the same instances will appear in the relative order implied by the `source_timestamp`.

Data sample

In addition to the sample sequence (`data_values`), the operation also returns a sequence of `DDS_SampleInfo` structures with the parameter `info_seq`. The `info_seq` structures and `data_values` also determine the behaviour of this operation.

Resource control

The initial (input) properties of the `data_values` and `info_seq` sequences determine the precise behaviour of the `SPACE_FooDataReader_read` operation. The sequences are modelled as having three properties: the current-length (`_length`), the maximum length (`_maximum`), and whether or not the sequence container owns the memory of the elements within (`_release`).

The initial (input) values of the `_length`, `_maximum`, and `_release` properties for the `data_values` and `info_seq` sequences govern the behaviour of the `SPACE_FooDataReader_read` operation as specified by the following rules:

- The values of `_length`, `_maximum`, and `_release` for the two sequences must be identical. Otherwise `SPACE_FooDataReader_read` returns `DDS_RETCODE_PRECONDITION_NOT_MET`
- On successful output, the values of `_length`, `_maximum`, and `_release` are the same for both sequences
- If the input `_maximum==0`, the `data_values` and `info_seq` sequences are filled with elements that are “loaned” by the `SPACE_FooDataReader`. On output, `_release` is `FALSE`, `_length` is set to the number of values returned, and `_maximum` is set to a value verifying `_maximum>=_length`. In this case the application will need to “return the loan” to the Data Distribution Service using the `SPACE_FooDataReader_return_loan` operation
- If the input `_maximum>0` and the input `_release==FALSE`, the `SPACE_FooDataReader_read` operation will fail and returns `DDS_RETCODE_PRECONDITION_NOT_MET`. This avoids the potential hard-to-detect memory leaks caused by an application forgetting to “return the loan”
- If input `_maximum>0` and the input `_release==TRUE`, the `SPACE_FooDataReader_read` operation will copy the `Foo` samples and `info_seq` values into the elements already inside the sequences. On output, `_release` is `TRUE`, `_length` is set to the number of values copied, and `_maximum` will remain unchanged. The application can control where the copy is placed and the application does not need to “return the loan”. The number of samples copied depends on the relative values of `_maximum` and `max_samples`:
 - If `_maximum==DDS_LENGTH_UNLIMITED`, at most `_maximum` values are copied. The use of this variant lets the application limit the number of samples returned to what the sequence can accommodate;
 - If `max_samples<=_maximum`, at most `max_samples` values are copied. The use of this variant lets the application limit the number of samples returned to fewer than what the sequence can accommodate;

- If `max_samples > _maximum`, the `SPACE_FooDataReader_read` operation will fail and returns `DDS_RETCODE_PRECONDITION_NOT_MET`. This avoids the potential confusion where the application expects to be able to access up to `max_samples`, but that number can never be returned, even if they are available in the `SPACE_FooDataReader`, because the output sequence cannot accommodate them.

Buffer Loan

As described above, upon return the `data_values` and `info_seq` sequences may contain elements “loaned” from the Data Distribution Service. If this is the case, the application will need to use the `SPACE_FooDataReader_return_loan` operation to return the “loan” once it is no longer using the data in the sequence. Upon return from `SPACE_FooDataReader_return_loan`, the sequence has `_maximum==0` and `_release==FALSE`.

The application can determine whether it is necessary to “return the loan” or not, based on the state of the sequences, when the `SPACE_FooDataReader_read` operation was called, or by accessing the “`_release`” property. However, in many cases it may be simpler to always call `SPACE_FooDataReader_return_loan`, as this operation is harmless (i.e. leaves all elements unchanged) if the sequence does not have a loan.

To avoid potential memory leaks, it is not allowed to change the length of the `data_values` and `info_seq` structures for which `_release==FALSE`. Furthermore, deleting a sequence for which `_release==FALSE` is considered to be an error except when the sequence is empty.

Data Sequence

On output, the sequence of data values and the sequence of `DDS_SampleInfo` structures are of the same length and are in an one-to-one correspondence. Each `DDS_SampleInfo` structures provides information, such as the `source_timestamp`, the `sample_state`, `view_state`, and `instance_state`, etc., about the matching sample.

Some elements in the returned sequence may not have valid data: the `valid_data` field in the `DDS_SampleInfo` indicates whether the corresponding data value contains any meaningful data. If not, the data value is just a ‘dummy’ sample for which only the keyfields have been assigned. It is used to accompany the `DDS_SampleInfo` that communicates a change in the `instance_state` of an instance for which there is no ‘real’ sample available.

For example, when an application always ‘takes’ all available samples of a particular instance, there is no sample available to report the disposal of that instance. In such a case the `DDS_DataReader` will insert a dummy sample into the `data_values` sequence to accompany the `DDS_SampleInfo` element in the `info_seq` sequence that communicates the disposal of the instance.

The act of reading a sample sets its `sample_state` to `DDS_READ_SAMPLE_STATE`. If the sample belongs to the most recent generation of the instance, it also sets the `view_state` of the instance to `DDS_NOT_NEW_VIEW_STATE`. It does not affect the `instance_state` of the instance.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - a sequence of data values is available.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - either or both of `data_values` or `info_seq` is an invalid pointer.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataReader` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one of the following is true:
 - the `max_samples > maximum` and `max_samples` is not `DDS_LENGTH_UNLIMITED`.
 - one or more values of `_length`, `_maximum`, and `_release` for the two sequences are not identical.
 - the `_maximum > 0` and the `_release == FALSE`.
- `DDS_RETCODE_NO_DATA` - no samples that meet the constraints are available.

3.5.2.62 `SPACE_FooDataReader_read_instance`

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataReader_read_instance
        (SPACE_FooDataReader _this,
         DDS_sequence_Foo *data_values,
```

```

DDS_SampleInfoSeq *info_seq,
const DDS_long max_samples,
const DDS_InstanceHandle_t a_handle,
const DDS_SampleStateMask sample_states,
const DDS_ViewStateMask view_states,
const DDS_InstanceStateMask instance_states);

```

Description

This operation reads a sequence of `Foo` samples of a single instance from the `SPACE_FooDataReader`.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

inout `DDS_sequence_Foo *data_values` - the returned sample data sequence. `data_values` is also used as an input to control the behaviour of this operation.

inout `DDS_SampleInfoSeq *info_seq` - the returned `DDS_SampleInfo` structure sequence. `info_seq` is also used as an input to control the behaviour of this operation.

in `const DDS_long max_samples` - the maximum number of samples that is returned.

in `const DDS_InstanceHandle_t a_handle` - the single instance, the samples belong to.

in `const DDS_SampleStateMask sample_states` - a mask, which selects only those samples with the desired sample states.

in `const DDS_ViewStateMask view_states` - a mask, which selects only those samples with the desired view states.

in `const DDS_InstanceStateMask instance_states` - a mask, which selects only those samples with the desired instance states.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED`, `DDS_RETCODE_PRECONDITION_NOT_MET` or `DDS_RETCODE_NO_DATA`.

Detailed Description

This operation reads a sequence of `Foo` samples of a single instance from the `SPACE_FooDataReader`. The behaviour is identical to `SPACE_FooDataReader_read` except for that all samples returned belong to the single specified instance whose handle is `a_handle`. Upon successful return, the data collection will contain samples all belonging to the same instance. The data is returned by the parameters `data_values` and `info_seq`. The corresponding `DDS_SampleInfo.instance_handle` in `info_seq` will have the value of `a_handle`. The `DDS_DataReader` will check that each sample belongs to the specified instance (indicated by `a_handle`) otherwise it will not place the sample in the returned collection.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - a sequence of data values is available.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - either or both of `data_values` or `info_seq` is an invalid pointer or `a_handle` is not a valid handle.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataReader` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one of the following is true:
 - the `max_samples>_maximum` and `max_samples` is not `DDS_LENGTH_UNLIMITED`.
 - one or more values of `_length`, `_maximum`, and `_release` for the two sequences are not identical.
 - the `_maximum>0` and the `_release==FALSE`.
- `DDS_RETCODE_NO_DATA` - no samples that meet the constraints are available.

3.5.2.63 `SPACE_FooDataReader_read_next_instance`

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataReader_read_next_instance
```

```
(SPACE_FooDataReader _this,
    DDS_sequence_Foo *data_values,
    DDS_SampleInfoSeq *info_seq,
    const DDS_long max_samples,
    const DDS_InstanceHandle_t a_handle,
    const DDS_SampleStateMask sample_states,
    const DDS_ViewStateMask view_states,
    const DDS_InstanceStateMask instance_states);
```

Description

This operation reads a sequence of `Foo` samples of the next single instance from the `SPACE_FooDataReader`.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

inout `DDS_sequence_Foo *data_values` - the returned sample data sequence. `data_values` is also used as an input to control the behaviour of this operation.

inout `DDS_SampleInfoSeq *info_seq` - the returned `DDS_SampleInfo` structure sequence. `info_seq` is also used as an input to control the behaviour of this operation.

in `const DDS_long max_samples` - the maximum number of samples that is returned.

in `const DDS_InstanceHandle_t a_handle` - the current single instance, the returned samples belong to the next single instance.

in `const DDS_SampleStateMask sample_states` - a mask, which selects only those samples with the desired sample states.

in `const DDS_ViewStateMask view_states` - a mask, which selects only those samples with the desired view states.

in `const DDS_InstanceStateMask instance_states` - a mask, which selects only those samples with the desired instance states.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED`, `DDS_RETCODE_PRECONDITION_NOT_MET` or `DDS_RETCODE_NO_DATA`.

Detailed Description

This operation reads a sequence of `Foo` samples of a single instance from the `SPACE_FooDataReader`. The behaviour is similar to `SPACE_FooDataReader_read_instance` (all samples returned belong to a single instance) except that the actual instance is not directly specified. Rather the samples will all belong to the ‘next’ instance with `instance_handle` ‘greater’ (according to some internal-defined order) than `a_handle`, that has available samples. The data is returned by the parameters `data_values` and `info_seq`. The corresponding `DDS_SampleInfo.instance_handle` in `info_seq` will have the value of the next instance with respect to `a_handle`.

Instance Order

The internal-defined order is not important and is implementation specific. The important thing is that, according to the Data Distribution Service, all instances are ordered relative to each other. This ordering is between the instances, that is, it does not depend on the actual samples received. For the purposes of this explanation it is ‘as if’ each instance handle was represented as a unique integer.

The behaviour of `SPACE_FooDataReader_read_next_instance` is ‘as if’ the `DDS_DataReader` invoked `SPACE_FooDataReader_read_instance` passing the smallest `instance_handle` among all the ones that:

- are greater than `a_handle`
- have available samples (i.e. samples that meet the constraints imposed by the specified states).

The special value `DDS_HANDLE_NIL` is guaranteed to be ‘less than’ any valid `instance_handle`. So the use of the parameter value `a_handle==DDS_HANDLE_NIL` will return the samples for the instance which has the smallest `instance_handle` among all the instances that contains available samples.

Typical use

The operation `SPACE_FooDataReader_read_next_instance` is intended to be used in an application-driven iteration where the application starts by passing `a_handle==DDS_HANDLE_NIL`, examines the samples returned, and then uses the `instance_handle` returned in the `DDS_SampleInfo` as the value of `a_handle` argument to the next call to `SPACE_FooDataReader_read_next_instance`. The iteration continues until `SPACE_FooDataReader_read_next_instance` returns the return value `DDS_RETCODE_NO_DATA`.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - either or both of *data_values* or *info_seq* is an invalid pointer or *a_handle* is not a valid handle.
- *DDS_RETCODE_ALREADY_DELETED* - the *SPACE_FooDataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataReader* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *max_samples*>*_maximum* and *max_samples* is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of *_length*, *_maximum*, and *_release* for the two sequences are not identical.
 - the *_maximum*>0 and the *_release*==FALSE.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.64 *SPACE_FooDataReader_read_next_instance_w_condition*

Synopsis

```
#include <Space.h>
    DDS_ReturnCode_t
    SPACE_FooDataReader_read_next_instance_w_condition
    (SPACE_FooDataReader _this,
     DDS_sequence_Foo *data_values,
     DDS_SampleInfoSeq *info_seq,
     const DDS_long max_samples,
     const DDS_InstanceHandle_t a_handle,
     const DDS_ReadCondition a_condition);
```

Description

This operation reads a sequence of *Foo* samples of the next single instance from the *SPACE_FooDataReader*.

Parameters

in SPACE_FooDataReader _this - the *SPACE_FooDataReader* object on which the operation is operated.

*inout DDS_sequence_Foo *data_values* - the returned sample data sequence. *data_values* is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned DDS_SampleInfo structure sequence. *info_seq* is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_InstanceHandle_t a_handle - the current single instance, the returned samples belong to the next single instance.

in const DDS_ReadCondition a_condition - a pointer to a DDS_ReadCondition or DDS_QueryCondition which filters the data before it is returned by the read operation.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are: DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of Foo samples of a single instance from the SPACE_FooDataReader, filtered by a DDS_ReadCondition or DDS_QueryCondition. The behaviour is identical to SPACE_FooDataReader_read_next_instance except for that the samples are filtered by a DDS_ReadCondition or DDS_QueryCondition. When using a DDS_ReadCondition, the result is the same as the SPACE_FooDataReader_read_next_instance operation with the same state parameters filled in as for the DDS_create_readcondition. In this way, the application can avoid repeating the same parameters, specified when creating the DDS_ReadCondition. When using a DDS_QueryCondition, a content based filtering can be done. When either using a DDS_ReadCondition or DDS_QueryCondition, the condition must be created by this SPACE_FooDataReader. Otherwise the operation will fail and returns DDS_RETCODE_PRECONDITION_NOT_MET.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - one or more of the *data_values*, or *info_seq* and *a_condition* parameters is an invalid pointer or *a_handle* is not a valid handle.
- *DDS_RETCODE_ALREADY_DELETED* - the *SPACE_FooDataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataReader* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *DDS_ReadCondition* or *DDS_QueryCondition* is not attached to this *SPACE_FooDataReader*.
 - the *max_samples>_maximum* and *max_samples* is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of *_length*, *_maximum*, and *_release* for the two sequences are not identical.
 - the *_maximum>0* and the *_release==FALSE*.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.65 *SPACE_FooDataReader_read_next_sample*

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataReader_read_next_sample
        (SPACE_FooDataReader _this,
         Foo *data_values,
         DDS_SampleInfo *sample_info);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.2.66 *SPACE_FooDataReader_read_w_condition*

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataReader_read_w_condition
        (SPACE_FooDataReader _this,
```

```

DDS_sequence_Foo *data_values,
DDS_SampleInfoSeq *info_seq,
const DDS_long max_samples,
const DDS_ReadCondition a_condition);

```

Description

This operation reads a sequence of Foo samples from the SPACE_FooDataReader, filtered by a DDS_ReadCondition or DDS_QueryCondition.

Parameters

in SPACE_FooDataReader _this - the SPACE_FooDataReader object on which the operation is operated.

*inout DDS_sequence_Foo *data_values* - the returned sample data sequence. data_values is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned DDS_SampleInfo structure sequence. info_seq is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_ReadCondition a_condition - a pointer to a DDS_ReadCondition or DDS_QueryCondition which filters the data before it is returned by the SPACE_FooDataReader_read operation.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of Foo samples from the SPACE_FooDataReader, filtered by a DDS_ReadCondition or DDS_QueryCondition. The condition pointer from both SPACE_FooDataReader_create_readcondition or SPACE_FooDataReader_create_querycondition may be used. The behaviour is identical to SPACE_FooDataReader_read except for that the samples are filtered by a DDS_ReadCondition or DDS_QueryCondition. When using a DDS_ReadCondition, the result is the same as the SPACE_FooDataReader_read operation with the same state parameters filled in

as for the `SPACE_FooDataReader_create_readcondition`. In this way, the application can avoid repeating the same parameters, specified when creating the `DDS_ReadCondition`. When using a `DDS_QueryCondition`, a content based filtering can be done. When either using a `DDS_ReadCondition` or `DDS_QueryCondition`, the condition must be created by this `SPACE_FooDataReader`. Otherwise the operation will fail and returns `DDS_RETCODE_PRECONDITION_NOT_MET`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - a sequence of data values is available.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - one or more of the `data_values`, or `info_seq` and `a_condition` parameters is an invalid pointer.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataReader` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one of the following is true:
 - the `DDS_ReadCondition` or `DDS_QueryCondition` is not attached to this `SPACE_FooDataReader`.
 - the `max_samples>_maximum` and `max_samples` is not `DDS_LENGTH_UNLIMITED`.
 - one or more values of `_length`, `_maximum`, and `_release` for the two sequences are not identical.
 - the `_maximum>0` and the `_release==FALSE`.
- `DDS_RETCODE_NO_DATA` - no samples that meet the constraints are available.

3.5.2.67 `SPACE_FooDataReader_return_loan`

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataReader_return_loan
        (SPACE_FooDataReader _this,
         DDS_sequence_Foo *data_values,
```

```
DDS_SampleInfoSeq *info_seq);
```

Description

This operation indicates to the `DDS_DataReader` that the application is done accessing the sequence of `data_values` and `info_seq`.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

inout `DDS_sequence_Foo *data_values` - the sample data sequence which was loaned from the `DDS_DataReader`.

inout `DDS_SampleInfoSeq *info_seq` - the `DDS_SampleInfo` structure sequence which was loaned from the `DDS_DataReader`.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED`, `DDS_RETCODE_OUT_OF_RESOURCES`, `DDS_RETCODE_NOT_ENABLED` or `DDS_RETCODE_PRECONDITION_NOT_MET`.

Detailed Description

This operation indicates to the `SPACE_FooDataReader` that the application is done accessing the sequence of `data_values` and `info_seq` obtained by some earlier invocation of the operation `SPACE_FooDataReader_read` or `SPACE_FooDataReader_take` (or any of the similar operations) on the `SPACE_FooDataReader`.

The `data_values` and `info_seq` must belong to a single related pair; that is, they should correspond to a pair returned from a single call to the operation `SPACE_FooDataReader_read` or `SPACE_FooDataReader_take`. The `data_values` and `info_seq` must also have been obtained from the same `DDS_DataReader` to which they are returned. If either of these conditions is not met the operation will fail and returns `DDS_RETCODE_PRECONDITION_NOT_MET`.

Buffer Loan

The operation `SPACE_FooDataReader_return_loan` allows implementations of the `SPACE_FooDataReader_read` and `SPACE_FooDataReader_take` operations to “loan” buffers from the Data Distribution Service to the application and in this manner provide “zero-copy” access to the data. During the loan, the Data Distribution Service will guarantee that the `data_values` and `info_seq` are not modified.

It is not necessary for an application to return the loans immediately after calling the operation `SPACE_FooDataReader_read` or `SPACE_FooDataReader_take`. However, as these buffers correspond to internal resources inside the `DDS_DataReader`, the application should not retain them indefinitely.

Calling `SPACE_FooDataReader_return_loan`

The use of the `SPACE_FooDataReader_return_loan` operation is only necessary if the call to the operation `SPACE_FooDataReader_read` or `SPACE_FooDataReader_take` “loaned” buffers to the application. This only occurs if the `data_values` and `info_seq` sequences had `_maximum=0` at the time the operation `SPACE_FooDataReader_read` or `SPACE_FooDataReader_take` was called. The application may also examine the ‘_release’ property of the collection to determine where there is an outstanding loan. However, calling the operation `SPACE_FooDataReader_return_loan` on a pair of sequences that does not have a loan is safe and has no side effects.

If the pair of sequences had a loan, upon return from the operation `SPACE_FooDataReader_return_loan` the pair of sequences has `_maximum=0`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the `DDS_DataReader` is informed that the sequences will not be used any more.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - either or both of `data_values` or `info_seq` is an invalid pointer.
- `DDS_RETCODE_ALREADY_DELETED` - the `SPACE_FooDataReader` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.
- `DDS_RETCODE_NOT_ENABLED` - the `SPACE_FooDataReader` is not enabled.
- `DDS_RETCODE_PRECONDITION_NOT_MET` - one of the following is true:
 - the `data_values` and `info_seq` do not belong to a single related pair.
 - the `data_values` and `info_seq` were not obtained from this `SPACE_FooDataReader`.

3.5.2.68 SPACE_FooDataReader_set_listener (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataReader` for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_set_listener
    (SPACE_FooDataReader _this,
     const struct DDS_DataReaderListener *a_listener,
     const DDS_StatusMask mask);
```

3.5.2.69 SPACE_FooDataReader_set_qos (inherited)

This operation is inherited and therefore not described here. See the class `DDS_DataReader` for further explanation.

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_set_qos
    (SPACE_FooDataReader _this,
     const DDS_DataReaderQos *qos);
```

3.5.2.70 SPACE_FooDataReader_take

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_take
    (SPACE_FooDataReader _this,
     DDS_sequence_Foo *data_values,
     DDS_SampleInfoSeq *info_seq,
     const DDS_long max_samples,
     const DDS_SampleStateMask sample_states,
     const DDS_ViewStateMask view_states,
     const DDS_InstanceStateMask instance_states);
```

Description

This operation reads a sequence of `Foo` samples from the `SPACE_FooDataReader` and by doing so, removes the data from the `SPACE_FooDataReader`.

Parameters

in `SPACE_FooDataReader _this` - the `SPACE_FooDataReader` object on which the operation is operated.

*inout DDS_sequence_Foo *data_values* - the returned sample data sequence. *data_values* is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned DDS_SampleInfo structure sequence. *info_seq* is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_SampleStateMask sample_states - a mask, which selects only those samples with the desired sample states.

in const DDS_ViewStateMask view_states - a mask, which selects only those samples with the desired view states.

in const DDS_InstanceStateMask instance_states - a mask, which selects only those samples with the desired instance states.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of *Foo* samples from the *SPACE_FooDataReader* and by doing so, removes the data from the *SPACE_FooDataReader*, so it can not be read or taken again. The behaviour is identical to *SPACE_FooDataReader_read* except for that the samples are removed from the *SPACE_FooDataReader*.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available and removed from the *SPACE_FooDataReader*.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - either or both of *data_values* or *info_seq* is an invalid pointer.

- *DDS_RETCODE_ALREADY_DELETED* - the *SPACE_FooDataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataReader* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *max_samples*>*_maximum* and *max_samples* is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of *_length*, *_maximum*, and *_release* for the two sequences are not identical.
 - the *_maximum*>0 and the *_release*==FALSE.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.71 *SPACE_FooDataReader_take_instance*

Synopsis

```
#include <Space.h>
    DDS_ReturnCode_t
    SPACE_FooDataReader_take_instance
    (SPACE_FooDataReader _this,
     DDS_sequence_Foo *data_values,
     DDS_SampleInfoSeq *info_seq,
     const DDS_long max_samples,
     const DDS_InstanceHandle_t a_handle,
     const DDS_SampleStateMask sample_states,
     const DDS_ViewStateMask view_states,
     const DDS_InstanceStateMask instance_states);
```

Description

This operation reads a sequence of *Foo* samples of a single instance from the *SPACE_FooDataReader* and by doing so, removes the data from the *SPACE_FooDataReader*.

Parameters

- in SPACE_FooDataReader _this* - the *SPACE_FooDataReader* object on which the operation is operated.
- inout DDS_sequence_Foo *data_values* - the returned sample data sequence. *data_values* is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned DDS_SampleInfo structure sequence. *info_seq* is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_InstanceHandle_t a_handle - the single instance, the samples belong to.

in const DDS_SampleStateMask sample_states - a mask, which selects only those samples with the desired sample states.

in const DDS_ViewStateMask view_states - a mask, which selects only those samples with the desired view states.

in const DDS_InstanceStateMask instance_states - a mask, which selects only those samples with the desired instance states.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of `Foo` samples of a single instance from the `SPACE_FooDataReader` and by doing so, removes the data from the `SPACE_FooDataReader`, so it can not be read or taken again. The behaviour is identical to `SPACE_FooDataReader_read_instance` except for that the samples are removed from the `SPACE_FooDataReader`.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available and removed from the `SPACE_FooDataReader`.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - either or both of *data_values* or *info_seq* is an invalid pointer or *a_handle* is not a valid handle.

- *DDS_RETCODE_ALREADY_DELETED* - the *SPACE_FooDataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataReader* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *max_samples*>*_maximum* and *max_samples* is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of *_length*, *_maximum*, and *_release* for the two sequences are not identical.
 - the *_maximum*>0 and the *_release*==FALSE.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.72 *SPACE_FooDataReader_take_next_instance*

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_take_next_instance
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_InstanceHandle_t a_handle,
 const DDS_SampleStateMask sample_states,
 const DDS_ViewStateMask view_states,
 const DDS_InstanceStateMask instance_states);
```

Description

This operation reads a sequence of *Foo* samples of the next single instance from the *SPACE_FooDataReader* and by doing so, removes the data from the *SPACE_FooDataReader*.

Parameters

- in SPACE_FooDataReader _this* - the *SPACE_FooDataReader* object on which the operation is operated.
- inout DDS_sequence_Foo *data_values* - the returned sample data sequence. *data_values* is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned DDS_SampleInfo structure sequence. *info_seq* is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_InstanceHandle_t a_handle - the current single instance, the returned samples belong to the next single instance.

in const DDS_SampleStateMask sample_states - a mask, which selects only those samples with the desired sample states.

in const DDS_ViewStateMask view_states - a mask, which selects only those samples with the desired view states.

in const DDS_InstanceStateMask instance_states - a mask, which selects only those samples with the desired instance states.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of `Foo` samples of a single instance from the `SPACE_FooDataReader` and by doing so, removes the data from the `SPACE_FooDataReader`, so it can not be read or taken again. The behaviour is identical to `SPACE_FooDataReader_read_next_instance` except for that the samples are removed from the `SPACE_FooDataReader`.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available and removed from the `SPACE_FooDataReader`.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - either or both of *data_values* or *info_seq* is an invalid pointer or *a_handle* is not a valid handle.

- *DDS_RETCODE_ALREADY_DELETED* - the *SPACE_FooDataReader* has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataReader* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *max_samples* > *_maximum* and *max_samples* is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of *_length*, *_maximum*, and *_release* for the two sequences are not identical.
 - the *_maximum* > 0 and the *_release* == FALSE.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.73 *SPACE_FooDataReader_take_next_instance_w_condition*

Synopsis

```
#include <Space.h>
    DDS_ReturnCode_t
    SPACE_FooDataReader_take_next_instance_w_condition
    (SPACE_FooDataReader _this,
     DDS_sequence_Foo *data_values,
     DDS_SampleInfoSeq *info_seq,
     const DDS_long max_samples,
     const DDS_InstanceHandle_t a_handle,
     const DDS_ReadCondition a_condition);
```

Description

This operation reads a sequence of *Foo* samples of the next single instance from the *SPACE_FooDataReader* and by doing so, removes the data from the *SPACE_FooDataReader*.

Parameters

in SPACE_FooDataReader _this - the *SPACE_FooDataReader* object on which the operation is operated.

*inout DDS_sequence_Foo *data_values* - the returned sample data sequence. *data_values* is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned *DDS_SampleInfo* structure sequence. *info_seq* is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_InstanceHandle_t a_handle - the current single instance, the returned samples belong to the next single instance.

in const DDS_ReadCondition a_condition - a pointer to a DDS_ReadCondition or DDS_QueryCondition which filters the data before it is returned by the read operation.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of `Foo` samples of a single instance from the `SPACE_FooDataReader`, filtered by a `DDS_ReadCondition` or `DDS_QueryCondition` and by doing so, removes the data from the `SPACE_FooDataReader`, so it can not be read or taken again. The behaviour is identical to `SPACE_FooDataReader_read_next_instance_w_condition` except for that the samples are removed from the `SPACE_FooDataReader`.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available and removed from the `SPACE_FooDataReader`.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - one or more of the `data_values`, `info_seq` and `a_condition` parameters is an invalid pointer or `a_handle` is not a valid handle.
- *DDS_RETCODE_ALREADY_DELETED* - the `SPACE_FooDataReader` has already been deleted.
- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the `SPACE_FooDataReader` is not enabled.

- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *DDS_ReadCondition* or *DDS_QueryCondition* is not attached to this *SPACE_FooDataReader*.
 - the *max_samples*>*_maximum* and *max_samples* is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of *_length*, *_maximum*, and *_release* for the two sequences are not identical.
 - the *_maximum*>0 and the *_release*==FALSE.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.74 *SPACE_FooDataReader_take_next_sample*

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_take_next_sample
(SPACE_FooDataReader _this,
 Foo *data_values,
 DDS_SampleInfo *sample_info);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.2.75 *SPACE_FooDataReader_take_w_condition*

Synopsis

```
#include <Space.h>
DDS_ReturnCode_t
SPACE_FooDataReader_take_w_condition
(SPACE_FooDataReader _this,
 DDS_sequence_Foo *data_values,
 DDS_SampleInfoSeq *info_seq,
 const DDS_long max_samples,
 const DDS_ReadCondition a_condition);
```

Description

This operation reads a sequence of *Foo* samples from the *SPACE_FooDataReader*, filtered by a *DDS_ReadCondition* or *DDS_QueryCondition* and by doing so, removes the data from the *SPACE_FooDataReader*.

Parameters

in SPACE_FooDataReader _this - the *SPACE_FooDataReader* object on which the operation is operated.

*inout DDS_sequence_Foo *data_values* - the returned sample data sequence. *data_values* is also used as an input to control the behaviour of this operation.

*inout DDS_SampleInfoSeq *info_seq* - the returned DDS_SampleInfo structure sequence. *info_seq* is also used as an input to control the behaviour of this operation.

in const DDS_long max_samples - the maximum number of samples that is returned.

in const DDS_ReadCondition a_condition - a pointer to a DDS_ReadCondition or DDS_QueryCondition which filters the data before it is returned by the SPACE_FooDataReader_read operation.

Return Value

DDS_ReturnCode_t - Possible return codes of the operation are:
 DDS_RETCODE_OK, DDS_RETCODE_ERROR, DDS_RETCODE_ILLEGAL_OPERATION, DDS_RETCODE_BAD_PARAMETER, DDS_RETCODE_ALREADY_DELETED, DDS_RETCODE_OUT_OF_RESOURCES, DDS_RETCODE_NOT_ENABLED, DDS_RETCODE_PRECONDITION_NOT_MET or DDS_RETCODE_NO_DATA.

Detailed Description

This operation reads a sequence of Foo samples from the SPACE_FooDataReader, filtered by a DDS_ReadCondition or DDS_QueryCondition and by doing so, removes the data from the SPACE_FooDataReader, so it can not be read or taken again. The behaviour is identical to SPACE_FooDataReader_read_w_condition except for that the samples are removed from the SPACE_FooDataReader.

Return Code

When the operation returns:

- *DDS_RETCODE_OK* - a sequence of data values is available and removed from the SPACE_FooDataReader.
- *DDS_RETCODE_ERROR* - an internal error has occurred.
- *DDS_RETCODE_ILLEGAL_OPERATION* - the operation is invoked on an inappropriate object.
- *DDS_RETCODE_BAD_PARAMETER* - one or more of the *data_values*, or *info_seq* and *a_condition* parameters is an invalid pointer.
- *DDS_RETCODE_ALREADY_DELETED* - the SPACE_FooDataReader has already been deleted.

- *DDS_RETCODE_OUT_OF_RESOURCES* - the Data Distribution Service ran out of resources to complete this operation.
- *DDS_RETCODE_NOT_ENABLED* - the *SPACE_FooDataReader* is not enabled.
- *DDS_RETCODE_PRECONDITION_NOT_MET* - one of the following is true:
 - the *DDS_ReadCondition* or *DDS_QueryCondition* is not attached to this *SPACE_FooDataReader*.
 - the `max_samples > maximum` and `max_samples` is not *DDS_LENGTH_UNLIMITED*.
 - one or more values of `_length`, `_maximum`, and `_release` for the two sequences are not identical.
 - the `_maximum > 0` and the `_release == FALSE`.
- *DDS_RETCODE_NO_DATA* - no samples that meet the constraints are available.

3.5.2.76 *SPACE_FooDataReader_wait_for_historical_data* (inherited)

This operation is inherited and therefore not described here. See the class *DDS_DataReader* for further explanation.

Synopsis

```
#include <Space.h>
        DDS_ReturnCode_t
        SPACE_FooDataReader_wait_for_historical_data
        (SPACE_FooDataReader _this,
         const DDS_Duration_t *max_wait);
```

3.5.3 Class *DDS_DataSample*

A *DDS_DataSample* represents an atom of data information (i.e. one value for an instance) as returned by the *DDS_DataReader*'s *DDS_DataReader_read*/*SPACE_FooDataReader_take* operations. It consists of two parts: A *DDS_SampleInfo* and the Data itself. The Data part is the data as produced by a *DDS_Publisher*. The *DDS_SampleInfo* part contains additional information related to the data provided by the Data Distribution Service.

3.5.4 Struct *DDS_SampleInfo*

The struct *DDS_SampleInfo* represents the additional information that accompanies the data in each sample that is read or taken.

The interface description of this struct is as follows:

```
struct DDS_SampleInfo
{
    DDS_SampleStateKind sample_state;
    DDS_ViewStateKind view_state;
    DDS_InstanceStateKind instance_state;
```

```

    DDS_Time_t source_timestamp;
    DDS_InstanceHandle_t instance_handle;
    DDS_InstanceHandle_t publication_handle;
    DDS_long disposed_generation_count;
    DDS_long no_writers_generation_count;
    DDS_long sample_rank;
    DDS_long generation_rank;
    DDS_long absolute_generation_rank;
    DDS_boolean valid_data;
};
/*
 * implemented API operations
 *     <no operations>
 */

```

The next paragraph describes the usage of the `DDS_SampleInfo` struct.

3.5.4.1 DDS_SampleInfo

Synopsis

```

#include <dds_dcps.h>
struct DDS_SampleInfo
{
    DDS_SampleStateKind sample_state;
    DDS_ViewStateKind view_state;
    DDS_InstanceStateKind instance_state;
    DDS_Time_t source_timestamp;
    DDS_InstanceHandle_t instance_handle;
    DDS_InstanceHandle_t publication_handle;
    DDS_long disposed_generation_count;
    DDS_long no_writers_generation_count;
    DDS_long sample_rank;
    DDS_long generation_rank;
    DDS_long absolute_generation_rank;
    DDS_boolean valid_data;
};

```

Description

The struct `DDS_SampleInfo` represents the additional information that accompanies the data in each sample that is read or taken.

Attributes

DDS_SampleStateKind sample_state - whether or not the corresponding data sample has already been read.

DDS_ViewStateKind view_state - whether the `DDS_DataReader` has already seen samples of the most-current generation of the related instance.

DDS_InstanceStateKind instance_state - whether the instance is alive, has no writers or is disposed of.

DDS_Time_t source_timestamp - the time provided by the DDS_DataWriter when the sample was written.

DDS_InstanceHandle_t instance_handle - the handle that identifies locally the corresponding instance.

DDS_InstanceHandle_t publication_handle - the handle that identifies locally the DDS_DataWriter that modified the instance. In fact it is an *instance_handle* of the built-in DCPSPublication sample that describes this DDS_DataWriter. It can be used as a parameter to the DDS_DataReader_get_matched_publication_data operation to obtain this built-in DCPSPublication sample.

DDS_long disposed_generation_count - the number of times the instance has become alive after it was disposed of explicitly by a DDS_DataWriter.

DDS_long no_writers_generation_count - the number of times the instance has become alive after it was disposed of because there were no DDS_DataWriter objects.

DDS_long sample_rank - the number of samples related to the same instance that are found in the collection returned by a DDS_DataReader_read or DDS_DataReader_take operation.

DDS_long generation_rank - the generation difference between the time the sample was received and the time the most recent sample in the collection was received.

DDS_long absolute_generation_rank - the generation difference between the time the sample was received and the time the most recent sample was received.

DDS_boolean valid_data - whether the DataSample contains any meaningful data. If not, the sample is only used to communicate a change in the *instance_state* of the instance.

Detailed Description

The struct `DDS_SampleInfo` represents the additional information that accompanies the data in each sample that is read or taken.

Sample Information

The struct `DDS_SampleInfo` represents the additional information that accompanies the data in each sample that is read or taken.

Generations

A generation is defined as: ‘the number of times an instance has become alive (with `instance_state==DDS_ALIVE_INSTANCE_STATE`) at the time the sample was received’. Note that the generation counters are initialized to zero when a Reader first detects a never-seen-before instance.

Two types of generations are distinguished: `disposed_generation_count` and `no_writers_generation_count`.

After a `DDS_DataWriter` disposes an instance, the `disposed_generation_count` for all Readers that already knew that instance will be incremented the next time the instance is written again.

If the `DDS_DataReader` detects that there are no live `DDS_DataWriter` entities, the `instance_state` of the `sample_info` will change from `DDS_ALIVE_INSTANCE_STATE` to `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`. The next time the instance is written, `no_writers_generation_count` will be incremented.

Sample Information

`DDS_SampleInfo` is the additional information that accompanies the data in each sample that is read or taken. It contains the following information:

- `sample_state` (`DDS_READ_SAMPLE_STATE` or `DDS_NOT_READ_SAMPLE_STATE`) indicates whether or not the corresponding data sample has already been read
- `view_state`, (`DDS_NEW_VIEW_STATE`, or `DDS_NOT_NEW_VIEW_STATE`) indicates whether the `DDS_DataReader` has already seen samples of the most-current generation of the related instance
- `instance_state` (`DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, or `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`) indicates whether the instance is alive, has no writers or if it has been disposed of:
 - `DDS_ALIVE_INSTANCE_STATE` if this instance is currently in existence
 - `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` if this instance was disposed of by a `DDS_DataWriter`
 - `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` none of the `DDS_DataWriter` objects currently “alive” (according to the `DDS_LivelinessQosPolicy`) are writing the instance.
- `source_timestamp` indicates the time provided by the `DDS_DataWriter` when the sample was written
- `instance_handle` indicates locally the corresponding instance

- `publication_handle` is used by the DDS implementation to locally identify the corresponding source DataWriter. You can access more detailed information about this particular publication by passing its `publication_handle` to either the `DDS_DataReader_get_matched_publication_data` operation or to the `DDS_PublicationBuiltinTopicDataReader_read_instance` operation on the built-in reader for the “DCPSPublication” topic.



Be aware that since `DDS_InstanceHandle_t` is an opaque datatype, it does not necessarily mean that the handle obtained from the `publication_handle` has the same value as the one that appears in the `instance_handle` field of the `DDS_SampleInfo` when retrieving the publication info through corresponding “DCPSPublication” built-in reader. You can’t just compare two handles to determine whether they represent the same publication. If you want to know whether two handles actually do represent the same publication, use both handles to retrieve their corresponding `DDS_PublicationBuiltinTopicData` samples and then compare the key field of both samples.

- `disposed_generation_count` indicates the number of times the instance has become alive after it was disposed of explicitly by a `DDS_DataWriter`, at the time the sample was received
- `no_writers_generation_count` indicates the number of times the instance has become alive after its `instance_state` has been `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`, at the time the sample was received
- `sample_rank` indicates the number of samples related to the same instance that follow in the collection returned by a `DDS_DataReader_read` or `DDS_DataReader_take` operation
- `generation_rank` indicates the generation difference (number of times the instance was disposed of and become alive again) between the time the sample was received and the time the most recent sample in the collection (related to the same instance) was received
- `absolute_generation_rank` indicates the generation difference (number of times the instance was disposed of and become alive again) between the time the sample was received and the time the most recent sample (which may not be in the returned collection), related to the same instance, was received.
- `valid_data` indicates whether the corresponding data value contains any meaningful data. If not, the data value is just a ‘dummy’ sample for which only the keyfields have been assigned. It is used to accompany the `DDS_SampleInfo` that communicates a change in the `instance_state` of an instance for which there is no ‘real’ sample available.

3.5.5 DDS_SubscriberListener Interface

Since a DDS_Subscriber is a kind of DDS_Entity, it has the ability to have a Listener associated with it. In this case, the associated Listener should be of type DDS_SubscriberListener. This interface must be implemented by the application. A user-defined class must be provided by the application which must extend from the DDS_SubscriberListener class. **All** DDS_SubscriberListener operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.



All operations for this interface must be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.

The DDS_SubscriberListener provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a QosPolicy setting, etc. The DDS_SubscriberListener is related to changes in communication status.

The interface description of this class is as follows:

```
/*
 * interface DDS_SubscriberListener
 */
/*
 * inherited from class DDS_DataReaderListener
 */
/* void
 *   DDS_SubscriberListener_on_requested_deadline_missed
 *   (void *listener_data,
 *    DDS_DataReader reader,
 *    const DDS_RequestedDeadlineMissedStatus *status);
 */
/* void
 *   DDS_SubscriberListener_on_requested_incompatible_qos
 *   (void *listener_data,
 *    DDS_DataReader reader,
 *    const DDS_RequestedIncompatibleQosStatus *status);
 */

/* void
 *   DDS_SubscriberListener_on_sample_rejected
 *   (void *listener_data,
 *    DDS_DataReader reader,
 *    const DDS_SampleRejectedStatus *status);
 */

/* void
```

```

*     DDS_SubscriberListener_on_liveliness_changed
*     (void *listener_data,
*       DDS_DataReader reader,
*       const DDS_LivelinessChangedStatus *status);
*/

/* void
*     DDS_SubscriberListener_on_data_available
*     (void *listener_data,
*       DDS_DataReader reader);
*/

/* void
*     DDS_SubscriberListener_on_subscription_matched
*     (void *listener_data,
*       DDS_DataReader reader,
*       const DDS_SubscriptionMatchedStatus *status);
*/

/* void
*     DDS_SubscriberListener_on_sample_lost
*     (void *listener_data,
*       DDS_DataReader reader,
*       const DDS_SampleLostStatus *status);
*/
/*
* abstract external operations
*/
void
    DDS_SubscriberListener_on_data_on_readers
    (void *listener_data,
     DDS_Subscriber subs);
/*
* implemented API operations
*/
DDS_SubscriberListener
    DDS_SubscriberListener__alloc
    (void);

```

The next paragraphs list all `DDS_SubscriberListener` operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited. The abstract operation is fully described since it must be implemented by the application.

3.5.5.1 `DDS_SubscriberListener__alloc`

Synopsis

```
#include <dds_dcps.h>
```



```
DDS_SubscriberListener
DDS_SubscriberListener__alloc
(void);
```

Description

This operation creates a new DDS_SubscriberListener.

Parameters

<none>

Return Value

DDS_SubscriberListener - Return value is the handle to the newly created DDS_SubscriberListener. In case of an error, a DDS_OBJECT_NIL pointer is returned.

Detailed Description

This operation creates a new DDS_SubscriberListener. The DDS_SubscriberListener must be created using this operation. In other words, the application is not allowed to declare an object of type DDS_SubscriberListener. When the application wants to release the DDS_SubscriberListener it must be released using DDS_free.

In case there are insufficient resources available to allocate the DDS_SubscriberListener, a DDS_OBJECT_NIL pointer is returned instead.

3.5.5.2 DDS_SubscriberListener_on_data_available (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataReaderListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
DDS_SubscriberListener_on_data_available
(void *listener_data,
DDS_DataReader reader);
```

3.5.5.3 DDS_SubscriberListener_on_data_on_readers (abstract)

Synopsis

```
#include <dds_dcps.h>
void
DDS_SubscriberListener_on_data_on_readers
(void *listener_data,
DDS_Subscriber subs);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when new data is available.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_Subscriber subs - contain a pointer to the DDS_Subscriber for which data is available (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when new data is available for this DDS_Subscriber. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant DDS_SubscriberListener is installed and enabled for the DDS_DATA_ON_READERS_STATUS.

The Data Distribution Service will provide a pointer to the DDS_Subscriber in the parameter subs for use by the application.

The statuses DDS_DATA_ON_READERS_STATUS and DDS_DATA_AVAILABLE_STATUS will occur together. In case these status changes occur, the Data Distribution Service will look for an attached and activated DDS_SubscriberListener or DDS_DomainParticipantListener (in that order) for the DDS_DATA_ON_READERS_STATUS. In case the DDS_DATA_ON_READERS_STATUS can not be handled, the Data Distribution Service will look for an attached and activated DDS_DataReaderListener, DDS_SubscriberListener or DDS_DomainParticipantListener for the DDS_DATA_AVAILABLE_STATUS (in that order).

Note that if DDS_SubscriberListener_on_data_on_readers is called, then the Data Distribution Service will not try to call DDS_SubscriberListener_on_data_available, however, the application can force a call to the callback function on_data_available of DDS_DataReaderListener objects that have data by means of the notify_datareaders operation.

3.5.5.4 DDS_SubscriberListener_on_liveliness_changed (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataReaderListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_SubscriberListener_on_liveliness_changed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_LivelinessChangedStatus *status);
```

3.5.5.5 DDS_SubscriberListener_on_requested_deadline_missed (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataReaderListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_SubscriberListener_on_requested_deadline_missed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedDeadlineMissedStatus *status);
```

3.5.5.6 DDS_SubscriberListener_on_requested_incompatible_qos (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataReaderListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_SubscriberListener_on_requested_incompatible_qos
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedIncompatibleQosStatus *status) =0;
```

3.5.5.7 DDS_SubscriberListener_on_sample_lost (inherited, abstract)

This operation is inherited and therefore not described here. See the class DDS_DataReaderListener for further explanation.

Synopsis

```
#include <dds_dcps.h>
```

```
void
    DDS_SubscriberListener_on_sample_lost
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleLostStatus *status);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.5.8 DDS_SubscriberListener_on_sample_rejected (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_SubscriberListener_on_sample_rejected
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleRejectedStatus *status);
```

3.5.5.9 DDS_SubscriberListener_on_subscription_matched (inherited, abstract)

This operation is inherited and therefore not described here. See the class `DDS_DataReaderListener` for further explanation.

Synopsis

```
#include <dds_dcps.h>
void
    DDS_SubscriberListener_on_subscription_matched
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SubscriptionMatchedStatus *status);
```

3.5.6 DDS_DataReaderListener interface

Since a `DDS_DataReader` is a kind of `DDS_Entity`, it has the ability to have a `Listener` associated with it. In this case, the associated `Listener` should be of type `DDS_DataReaderListener`. This interface must be implemented by the application. A user-defined class must be provided by the application which must extend from the `DDS_DataReaderListener` class. **All** `DDS_DataReaderListener` operations **must** be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.



All operations for this interface must be implemented in the user-defined class, it is up to the application whether an operation is empty or contains some functionality.

The `DDS_DataReaderListener` provides a generic mechanism (actually a callback function) for the Data Distribution Service to notify the application of relevant asynchronous status change events, such as a missed deadline, violation of a `QosPolicy` setting, etc. The `DDS_DataReaderListener` is related to changes in communication status.

The interface description of this class is as follows:

```

/*
 * interface DDS_DataReaderListener
 */
/*
 * abstract external operations
 */
void
    DDS_DataReaderListener_on_requested_deadline_missed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedDeadlineMissedStatus *status);
void
    DDS_DataReaderListener_on_requested_incompatible_qos
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedIncompatibleQosStatus *status);

void
    DDS_DataReaderListener_on_sample_rejected
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleRejectedStatus *status);

void
    DDS_DataReaderListener_on_liveliness_changed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_LivelinessChangedStatus *status);

void
    DDS_DataReaderListener_on_data_available
        (void *listener_data,
         DDS_DataReader reader);

void
    DDS_DataReaderListener_on_subscription_matched
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SubscriptionMatchedStatus *status);

void
    DDS_DataReaderListener_on_sample_lost

```

```

        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleLostStatus *status);
/*
 * implemented API operations
 */
DDS_DataReaderListener
DDS_DataReaderListener__alloc
(void);

```

The next paragraphs describe the usage of all `DDS_DataReaderListener` operations. These abstract operations are fully described because they must be implemented by the application.

3.5.6.1 `DDS_DataReaderListener__alloc`

Synopsis

```

#include <dds_dcps.h>
DDS_DataReaderListener
DDS_DataReaderListener__alloc
(void);

```

Description

This operation creates a new `DDS_DataReaderListener`.

Parameters

<none>

Return Value

DDS_DataReaderListener - Return value is the handle to the newly created `DDS_DataReaderListener`. In case of an error, a `DDS_OBJECT_NIL` pointer is returned.

Detailed Description

This operation creates a new `DDS_DataReaderListener`. The `DDS_DataReaderListener` must be created using this operation. In other words, the application is not allowed to declare an object of type `DDS_DataReaderListener`. When the application wants to release the `DDS_DataReaderListener` it must be released using `DDS_free`.

In case there are insufficient resources available to allocate the `DDS_DataReaderListener`, a `DDS_OBJECT_NIL` pointer is returned instead.

3.5.6.2 DDS_DataReaderListener_on_data_available (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_data_available
        (void *listener_data,
         DDS_DataReader reader);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when new data is available.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataReader reader - contain a pointer to the DDS_DataReader for which data is available (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when new data is available for this DDS_DataReader. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant DDS_DataReaderListener is installed and enabled for the DDS_DATA_AVAILABLE_STATUS.

The Data Distribution Service will provide a pointer to the DDS_DataReader in the parameter reader for use by the application.

The statuses DDS_DATA_ON_READERS_STATUS and DDS_DATA_AVAILABLE_STATUS will occur together. In case these status changes occur, the Data Distribution Service will look for an attached and activated DDS_SubscriberListener or DDS_DomainParticipantListener (in that order) for the DDS_DATA_ON_READERS_STATUS. In case the DDS_DATA_ON_READERS_STATUS can not be handled, the Data Distribution Service will look for an attached and activated DDS_DataReaderListener, DDS_SubscriberListener or DDS_DomainParticipantListener for the DDS_DATA_AVAILABLE_STATUS (in that order).

Note that if `DDS_SubscriberListener_on_data_on_readers` is called, then the Data Distribution Service will not try to call `DDS_DataReaderListener_on_data_available`, however, the application can force a call to the `DDS_DataReader` objects that have data by means of the `DDS_Subscriber_notify_datareaders` operation.

3.5.6.3 DDS_DataReaderListener_on_liveliness_changed (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_liveliness_changed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_LivelinessChangedStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the liveliness of one or more `DDS_DataWriter` objects that were writing instances read through this `DDS_DataReader` has changed.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataReader reader - contain a pointer to the `DDS_DataReader` for which the liveliness of one or more `DDS_DataWriter` objects has changed (this is an input to the application provided by the Data Distribution Service).

*in const DDS_LivelinessChangedStatus *status* - contain the `DDS_LivelinessChangedStatus` struct (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the liveliness of one or more `DDS_DataWriter` objects that were writing instances read through this `DDS_DataReader` has changed. In other words, some `DDS_DataWriter` have become “alive” or “not alive”. The implementation may be left empty when this

functionality is not needed. This operation will only be called when the relevant `DDS_DataReaderListener` is installed and enabled for the `DDS_LIVELINESS_CHANGED_STATUS`.

The Data Distribution Service will provide a pointer to the `DDS_DataReader` in the parameter `reader` and the `DDS_LivelinessChangedStatus` struct for use by the application.

3.5.6.4 `DDS_DataReaderListener_on_requested_deadline_missed` (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_requested_deadline_missed
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedDeadlineMissedStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the deadline that the `DDS_DataReader` was expecting through its `DDS_DeadlineQosPolicy` was not respected.

Parameters

inout `void *listener_data` - a pointer to a user-defined object which may be used for identification of the Listener.

in `DDS_DataReader reader` - contain a pointer to the `DDS_DataReader` for which the deadline was missed (this is an input to the application provided by the Data Distribution Service).

in `const DDS_RequestedDeadlineMissedStatus *status` - contain the `DDS_RequestedDeadlineMissedStatus` struct (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the deadline that the `DDS_DataReader` was expecting through its `DDS_DeadlineQosPolicy` was not respected for a specific instance. The implementation may be left empty

when this functionality is not needed. This operation will only be called when the relevant `DDS_DataReaderListener` is installed and enabled for the `DDS_REQUESTED_DEADLINE_MISSED_STATUS`.

The Data Distribution Service will provide a pointer to the `DDS_DataReader` in the parameter `reader` and the `DDS_RequestedDeadlineMissedStatus` struct in the parameter `status` for use by the application.

3.5.6.5 `DDS_DataReaderListener_on_requested_incompatible_qos` (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_requested_incompatible_qos
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_RequestedIncompatibleQosStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` changes.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataReader reader - a pointer to the `DDS_DataReader` provided by the Data Distribution Service.

*in const DDS_RequestedIncompatibleQosStatus *status* - the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` struct provided by the Data Distribution Service.

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` changes. The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant `DDS_DataReaderListener` is installed and enabled for the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS`.

The Data Distribution Service will provide a pointer to the `DDS_DataReader` in the parameter `reader` and the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` struct in the parameter `status`, for use by the application.

The application can use this operation as a callback function implementing a proper response to the status change. This operation is enabled by setting the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS` in the mask in the call to `DDS_DataReader_set_listener`. When the `DDS_DataReaderListener` on the `DDS_DataReader` is not enabled for the `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS`, the status change will propagate to the `DDS_SubscriberListener` of the `DDS_Subscriber` (if enabled) or to the `DDS_DomainParticipantListener` of the `DDS_DomainParticipant` (if enabled).

3.5.6.6 DDS_DataReaderListener_on_sample_lost (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_sample_lost
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleLostStatus *status);
```

NOTE: This operation is not yet implemented. It is scheduled for a future release.

3.5.6.7 DDS_DataReaderListener_on_sample_rejected (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_sample_rejected
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SampleRejectedStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when a sample has been rejected.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataReader reader - contain a pointer to the DDS_DataReader for which a sample has been rejected (this is an input to the application provided by the Data Distribution Service).

*in const DDS_SampleRejectedStatus *status* - contain the DDS_SampleRejectedStatus struct (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation is the external operation (interface, which must be implemented by the application) that is called by the Data Distribution Service when a (received) sample has been rejected. Samples may be rejected by the DDS_DataReader when it runs out of `resource_limits` to store incoming samples. Usually this means that old samples need to be ‘consumed’ (for example by ‘taking’ them instead of ‘reading’ them) to make room for newly incoming samples.

The implementation may be left empty when this functionality is not needed. This operation will only be called when the relevant DDS_DataReaderListener is installed and enabled for the DDS_SAMPLE_REJECTED_STATUS.

The Data Distribution Service will provide a pointer to the DDS_DataReader in the parameter `reader` and the DDS_SampleRejectedStatus struct in the parameter `status` for use by the application.

3.5.6.8 DDS_DataReaderListener_on_subscription_matched (abstract)

Synopsis

```
#include <dds_dcps.h>
void
    DDS_DataReaderListener_on_subscription_matched
        (void *listener_data,
         DDS_DataReader reader,
         const DDS_SubscriptionMatchedStatus *status);
```

Description

This operation must be implemented by the application and is called by the Data Distribution Service when a new match has been discovered for the current subscription, or when an existing match has ceased to exist.

Parameters

*inout void *listener_data* - a pointer to a user-defined object which may be used for identification of the Listener.

in DDS_DataReader reader - contains a pointer to the DDS_DataReader for which a match has been discovered (this is an input to the application provided by the Data Distribution Service).

*in const SubscriptionMatchedStatus *status* - contains the SubscriptionMatchedStatus struct (this is an input to the application provided by the Data Distribution Service).

Return Value

<none>

Detailed Description

This operation must be implemented by the application and is called by the Data Distribution Service when a new match has been discovered for the current subscription, or when an existing match has ceased to exist. Usually this means that a new DataWriter that matches the Topic and that has compatible Qos as the current DDS_DataReader has either been discovered, or that a previously discovered DataWriter has ceased to be matched to the current DDS_DataReader. A DataWriter may cease to match when it gets deleted, when it changes its Qos to a value that is incompatible with the current DDS_DataReader or when either the DDS_DataReader or the DataWriter has chosen to put its matching counterpart on its ignore-list using the DDS_DomainParticipant_ignore_publication or DDS_DomainParticipant_ignore_subscription operations.

The implementation of this Listener operation may be left empty when this functionality is not needed: it will only be called when the relevant DDS_DataReaderListener is installed and enabled for the DDS_SUBSCRIPTION_MATCHED_STATUS.

The Data Distribution Service will provide a pointer to the DDS_DataReader in the parameter reader and the DDS_SubscriptionMatchedStatus struct in the parameter status for use by the application.

3.5.7 Class DDS_ReadCondition

The DDS_DataReader objects can create a set of DDS_ReadCondition (and DDS_StatusCondition) objects which provide support (in conjunction with DDS_WaitSet objects) for an alternative communication style between the Data Distribution Service and the application (i.e., wait-based rather than notification-based).

DDS_ReadCondition objects allow an DDS_DataReader to specify the data samples it is interested in (by specifying the desired sample-states, view-states, and instance-states); see the parameter definitions for DDS_DataReader's DDS_DataReader_create_readcondition operation. This allows the Data Distribution Service to trigger the condition only when suitable information is

available. DDS_ReadCondition objects are to be used in conjunction with a DDS_WaitSet. More than one DDS_ReadCondition may be attached to the same DDS_DataReader.

The interface description of this class is as follows:

```

/*
 * interface DDS_ReadCondition
 */
/*
 * inherited from DDS_Condition
 */
/* DDS_boolean
 *   DDS_ReadCondition_get_trigger_value
 *   (DDS_ReadCondition _this);
 */
/*
 * implemented API operations
 */
    DDS_SampleStateMask
        DDS_ReadCondition_get_sample_state_mask
            (DDS_ReadCondition _this);

    DDS_ViewStateMask
        DDS_ReadCondition_get_view_state_mask
            (DDS_ReadCondition _this);

    DDS_InstanceStateMask
        DDS_ReadCondition_get_instance_state_mask
            (DDS_ReadCondition _this);

    DDS_DataReader
        DDS_ReadCondition_get_datareader
            (DDS_ReadCondition _this);

```

The next paragraphs describe the usage of all DDS_ReadCondition operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.5.7.1 DDS_ReadCondition_get_datareader

Synopsis

```

#include <dds_dcps.h>
DDS_DataReader
    DDS_ReadCondition_get_datareader
        (DDS_ReadCondition _this);

```

Description

This operation returns the `DDS_DataReader` associated with the `DDS_ReadCondition`.

Parameters

in `DDS_ReadCondition _this` - the `DDS_ReadCondition` object on which the operation is operated.

Return Value

`DDS_DataReader` - Result value is a pointer to the `DDS_DataReader`.

Detailed Description

This operation returns the `DDS_DataReader` associated with the `DDS_ReadCondition`. Note that there is exactly one `DDS_DataReader` associated with each `DDS_ReadCondition` (*i.e.* the `DDS_DataReader` that created the `DDS_ReadCondition` object).

3.5.7.2 DDS_ReadCondition_get_instance_state_mask

Synopsis

```
#include <dds_dcps.h>
DDS_InstanceStateMask
    DDS_ReadCondition_get_instance_state_mask
        (DDS_ReadCondition _this);
```

Description

This operation returns the set of `instance_states` that are taken into account to determine the `trigger_value` of the `DDS_ReadCondition`.

Parameters

in `DDS_ReadCondition _this` - the `DDS_ReadCondition` object on which the operation is operated.

Return Value

`DDS_InstanceStateMask` - Result value are the `instance_states` specified when the `DDS_ReadCondition` was created.

Detailed Description

This operation returns the set of `instance_states` that are taken into account to determine the `trigger_value` of the `DDS_ReadCondition`.

The `instance_states` returned are the `instance_states` specified when the `DDS_ReadCondition` was created. `instance_states` can be `DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` or a combination of these.

3.5.7.3 `DDS_ReadCondition_get_sample_state_mask`

Synopsis

```
#include <dds_dcps.h>
DDS_SampleStateMask
    DDS_ReadCondition_get_sample_state_mask
        (DDS_ReadCondition _this);
```

Description

This operation returns the set of `sample_states` that are taken into account to determine the `trigger_value` of the `DDS_ReadCondition`.

Parameters

in `DDS_ReadCondition _this` - the `DDS_ReadCondition` object on which the operation is operated.

Return Value

`DDS_SampleStateMask` - Result value are the `sample_states` specified when the `DDS_ReadCondition` was created.

Detailed Description

This operation returns the set of `sample_states` that are taken into account to determine the `trigger_value` of the `DDS_ReadCondition`.

The `sample_states` returned are the `sample_states` specified when the `DDS_ReadCondition` was created. `sample_states` can be `DDS_READ_SAMPLE_STATE`, `DDS_NOT_READ_SAMPLE_STATE` or both.

3.5.7.4 `DDS_ReadCondition_get_trigger_value` (inherited)

This operation is inherited and therefore not described here. See the class `DDS_Condition` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_boolean
    DDS_ReadCondition_get_trigger_value
        (DDS_ReadCondition _this);
```


3.5.7.5 DDS_ReadCondition_get_view_state_mask

Synopsis

```
#include <dds_dcps.h>
DDS_ViewStateMask
    DDS_ReadCondition_get_view_state_mask
        (DDS_ReadCondition _this);
```

Description

This operation returns the set of `view_states` that are taken into account to determine the `trigger_value` of the `DDS_ReadCondition`.

Parameters

in `DDS_ReadCondition _this` - the `DDS_ReadCondition` object on which the operation is operated.

Return Value

`DDS_ViewStateMask` - Result value are the `view_states` specified when the `DDS_ReadCondition` was created.

Detailed Description

This operation returns the set of `view_states` that are taken into account to determine the `trigger_value` of the `DDS_ReadCondition`.

The `view_states` returned are the `view_states` specified when the `DDS_ReadCondition` was created. `view_states` can be `DDS_NEW_VIEW_STATE`, `DDS_NOT_NEW_VIEW_STATE` or both.

3.5.8 Class DDS_QueryCondition

`DDS_QueryCondition` objects are specialized `DDS_ReadCondition` objects that allow the application to specify a filter on the locally available data. The `DDS_DataReader` objects accept a set of `DDS_QueryCondition` objects for the `DDS_DataReader` and provide support (in conjunction with `DDS_WaitSet` objects) for an alternative communication style between the Data Distribution Service and the application (i.e., wait-based rather than notification-based).

Query Function

`DDS_QueryCondition` objects allow an application to specify the data samples it is interested in (by specifying the desired sample-states, view-states, instance-states and query expression); see the parameter definitions for `DDS_DataReader's` `DDS_DataReader_read/DDS_DataReader_take` operations. This allows the Data Distribution Service to trigger the condition only when suitable information is

available. DDS_QueryCondition objects are to be used in conjunction with a DDS_WaitSet. More than one DDS_QueryCondition may be attached to the same DDS_DataReader.

The query (query_expression) is similar to an SQL WHERE clause and can be parameterized by arguments that are dynamically changeable with the DDS_QueryCondition_set_query_arguments operation.

The interface description of this class is as follows:

```

/*
 * interface DDS_QueryCondition
 */
/*
 * inherited from DDS_ReadCondition
 */
/* DDS_SampleStateMask
 *   DDS_QueryCondition_get_sample_state_mask
 *   (DDS_QueryCondition _this);
 */
/* DDS_ViewStateMask
 *   DDS_QueryCondition_get_view_state_mask
 *   (DDS_QueryCondition _this);
 */

/* DDS_InstanceStateMask
 *   DDS_QueryCondition_get_instance_state_mask
 *   (DDS_QueryCondition _this);
 */

/* DDS_DataReader
 *   DDS_QueryCondition_get_datareader
 *   (DDS_QueryCondition _this);
 */
/* DDS_boolean
 *   DDS_QueryCondition_get_trigger_value
 *   (DDS_QueryCondition _this);
 */
/*
 * implemented API operations
 */
    DDS_string
        DDS_QueryCondition_get_query_expression
        (DDS_QueryCondition _this);

    DDS_ReturnCode_t
        DDS_QueryCondition_get_query_parameters
        (DDS_QueryCondition _this,
         DDS_StringSeq *query_parameters);

```

```

DDS_ReturnCode_t
    DDS_QueryCondition_set_query_parameters
        (DDS_QueryCondition _this,
         const DDS_StringSeq *query_parameters);

```

The next paragraphs describe the usage of all DDS_QueryCondition operations. The inherited operations are listed but not fully described because they are not implemented in this class. The full description of these operations is given in the classes from which they are inherited.

3.5.8.1 DDS_QueryCondition_get_datareader (inherited)

This operation is inherited and therefore not described here. See the class DDS_ReadCondition for further explanation.

Synopsis

```

#include <dds_dcps.h>
DDS_DataReader
    DDS_QueryCondition_get_datareader
        (DDS_QueryCondition _this);

```

3.5.8.2 DDS_QueryCondition_get_instance_state_mask (inherited)

This operation is inherited and therefore not described here. See the class DDS_ReadCondition for further explanation.

Synopsis

```

#include <dds_dcps.h>
DDS_InstanceStateMask
    DDS_QueryCondition_get_instance_state_mask
        (DDS_QueryCondition _this);

```

3.5.8.3 DDS_QueryCondition_get_query_parameters

Synopsis

```

#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_QueryCondition_get_query_parameters
        (DDS_QueryCondition _this,
         DDS_StringSeq *query_parameters);

```

Description

This operation returns the `query_parameters` associated with the DDS_QueryCondition.

Parameters

in `DDS_QueryCondition _this` - the `DDS_QueryCondition` object on which the operation is operated.

inout `DDS_StringSeq *query_parameters` - a handle to a sequence of strings that will be used to store the parameters used in the SQL expression.

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation obtains the `query_parameters` associated with the `DDS_QueryCondition`. That is, the parameters specified on the last successful call to `DDS_QueryCondition_set_query_arguments` or, if `DDS_QueryCondition_set_query_arguments` was never called, the arguments specified when the `DDS_QueryCondition` were created. The resulting handle contains a sequence of strings with the parameters used in the SQL expression (i.e., the `%n` tokens in the expression). The number of parameters in the result sequence will exactly match the number of `%n` tokens in the query expression associated with the `DDS_QueryCondition`.

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the existing set of query parameters applied to this `DDS_QueryCondition` has successfully been copied into the specified `query_parameters` parameter.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_QueryCondition` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

3.5.8.4 `DDS_QueryCondition_get_query_expression`

Synopsis

```
#include <dds_dcps.h>
```

```
DDS_string
    DDS_QueryCondition_get_query_expression
        (DDS_QueryCondition _this);
```

Description

This operation returns the query expression associated with the DDS_QueryCondition.

Parameters

in DDS_QueryCondition _this - the DDS_QueryCondition object on which the operation is operated.

Return Value

DDS_string - Result value is a pointer to the query expression associated with the DDS_QueryCondition.

Detailed Description

This operation returns the query expression associated with the DDS_QueryCondition. That is, the expression specified when the DDS_QueryCondition was created. The operation will return DDS_OBJECT_NIL when there was an internal error or when the DDS_QueryCondition was already deleted. If there were no parameters, an empty sequence is returned.

It is the applications responsibility to free the allocated memory for the DDS_StringSeq.

3.5.8.5 DDS_QueryCondition_get_sample_state_mask (inherited)

This operation is inherited and therefore not described here. See the class DDS_ReadCondition for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_SampleStateMask
    DDS_QueryCondition_get_sample_state_mask
        (DDS_QueryCondition _this);
```

3.5.8.6 DDS_QueryCondition_get_trigger_value (inherited)

This operation is inherited and therefore not described here. See the class DDS_ReadCondition for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_boolean
    DDS_QueryCondition_get_trigger_value
```

```
(DDS_QueryCondition _this);
```

3.5.8.7 DDS_QueryCondition_get_view_state_mask (inherited)

This operation is inherited and therefore not described here. See the class `DDS_ReadCondition` for further explanation.

Synopsis

```
#include <dds_dcps.h>
DDS_ViewStateMask
    DDS_QueryCondition_get_view_state_mask
        (DDS_QueryCondition _this);
```

3.5.8.8 DDS_QueryCondition_set_query_parameters

Synopsis

```
#include <dds_dcps.h>
DDS_ReturnCode_t
    DDS_QueryCondition_set_query_parameters
        (DDS_QueryCondition _this,
         const DDS_StringSeq *query_parameters);
```

Description

This operation changes the query parameters associated with the `DDS_QueryCondition`.

Parameters

in `DDS_QueryCondition _this` - the `DDS_QueryCondition` object on which the operation is operated.

in `const DDS_StringSeq *query_parameters` - a sequence of strings which are the parameters used in the SQL query string (i.e., the “%n” tokens in the expression).

Return Value

`DDS_ReturnCode_t` - Possible return codes of the operation are:
`DDS_RETCODE_OK`, `DDS_RETCODE_ERROR`, `DDS_RETCODE_ILLEGAL_OPERATION`, `DDS_RETCODE_BAD_PARAMETER`, `DDS_RETCODE_ALREADY_DELETED` or `DDS_RETCODE_OUT_OF_RESOURCES`.

Detailed Description

This operation changes the query parameters associated with the `DDS_QueryCondition`. The parameter `query_parameters` is a sequence of strings which are the parameters values used in the SQL query string (i.e., the “%n” tokens in the expression). The number of values in `query_parameters` must be

equal or greater than the highest referenced %n token in the `query_expression` (e.g. if %1 and %8 are used as parameter in the `query_expression`, the `query_parameters` should at least contain `n+1 = 9` values).

Return Code

When the operation returns:

- `DDS_RETCODE_OK` - the query parameters associated with the `DDS_QueryCondition` are changed.
- `DDS_RETCODE_ERROR` - an internal error has occurred.
- `DDS_RETCODE_ILLEGAL_OPERATION` - the operation is invoked on an inappropriate object.
- `DDS_RETCODE_BAD_PARAMETER` - the number of parameters in `query_parameters` does not match the number of “%n” tokens in the expression for this `DDS_QueryCondition` or one of the parameters is an illegal parameter.
- `DDS_RETCODE_ALREADY_DELETED` - the `DDS_QueryCondition` has already been deleted.
- `DDS_RETCODE_OUT_OF_RESOURCES` - the Data Distribution Service ran out of resources to complete this operation.

A close-up, low-angle photograph of a computer keyboard, focusing on the central and lower-right keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

APPENDICES

A

Quality Of Service

Each `DDS_Entity` is accompanied by an `<DDS_Entity>Qos` structure that implements the basic mechanism for an application to specify Quality of Service attributes. This structure consists of `DDS_Entity` specific `QosPolicy` attributes. `QosPolicy` attributes are structured types where each type specifies the information that controls an `DDS_Entity` related (configurable) attribute of the Data Distribution Service.

Affected Entities

Each `DDS_Entity` can be configured with a set of `QosPolicy` settings. However, any `DDS_Entity` cannot support any `QosPolicy`. For instance, a `DDS_DomainParticipant` supports different `QosPolicy` settings than a `DDS_Topic` or a `DDS_Publisher`. The set of `QosPolicy` settings is implemented as a struct of `QosPolicy` structs, identified as `<DDS_Entity>Qos`. Each `<DDS_Entity>Qos` struct only contains those `QosPolicy` structs relevant to the specific `DDS_Entity`. The `<DDS_Entity>Qos` struct serves as the parameter to operations which require a `Qos`. `<DDS_Entity>Qos` struct is the API implementation of the QoS. Depending on the specific `<DDS_Entity>Qos`, it controls the behaviour of a `DDS_Topic`, `DDS_DataWriter`, `DDS_DataReader`, `DDS_Publisher`, `DDS_Subscriber`, `DDS_DomainParticipant` or `DDS_DomainParticipantFactory`¹.

Basic Usage

The basic way to modify or set the `<DDS_Entity>Qos` is by using an `DDS_<Entity>_get_qos` operation to get all `QosPolicy` settings from this `DDS_Entity` (that is the `<DDS_Entity>Qos`), modify several specific `QosPolicy` settings and put them back using an `DDS_<DDS_Entity>_set_qos` operation to set all `QosPolicy` settings on this `DDS_Entity` (that is the `<DDS_Entity>Qos`). An example of these operations for the `DDS_DataWriterQos` are

-
1. Note that the `DDS_DomainParticipantFactory` is a special kind of entity: it does not inherit from `DDS_Entity`, nor does it have a `DDS_Listener` or `DDS_StatusCondition`, but its behaviour can be controlled by its own set of `QosPolicies`.

DDS_Publisher_get_default_datawriter_qos and DDS_Publisher_set_default_datawriter_qos, which take the DDS_DataWriterQos as a parameter.

The interface description of this struct is as follows:

```

/*
 * struct <name>QosPolicy
 *     see appendix
 */
/*
 * struct <DDS_Entity>Qos
 */
struct DDS_DomainParticipantFactoryQos
{
    DDS_EntityFactoryQosPolicy    entity_factory; };
struct DDS_DomainParticipantQos
{
    DDS_UserDataQosPolicy          user_data;
    DDS_EntityFactoryQosPolicy     entity_factory;
    DDS_SchedulingQosPolicy        watchdog_scheduling;
    DDS_SchedulingQosPolicy        listener_scheduling; };
struct DDS_TopicQos
{
    DDS_TopicDataQosPolicy         topic_data;
    DDS_DurabilityQosPolicy        durability;
    DDS_DurabilityServiceQosPolicy durability_service;
    DDS_DeadlineQosPolicy          deadline;
    DDS_LatencyBudgetQosPolicy     latency_budget;
    DDS_LivelinessQosPolicy        liveliness;
    DDS_ReliabilityQosPolicy        reliability;
    DDS_DestinationOrderQosPolicy  destination_order;
    DDS_HistoryQosPolicy           history;
    DDS_ResourceLimitsQosPolicy    resource_limits;
    DDS_TransportPriorityQosPolicy transport_priority;
    DDS_LifespanQosPolicy          lifespan;
    DDS_OwnershipQosPolicy         ownership; };
struct DDS_DataWriterQos
{
    DDS_DurabilityQosPolicy        durability;
    DDS_DeadlineQosPolicy          deadline;
    DDS_LatencyBudgetQosPolicy     latency_budget;
    DDS_LivelinessQosPolicy        liveliness;
    DDS_ReliabilityQosPolicy        reliability;
    DDS_DestinationOrderQosPolicy  destination_order;
    DDS_HistoryQosPolicy           history;
    DDS_ResourceLimitsQosPolicy    resource_limits;
    DDS_TransportPriorityQosPolicy transport_priority;
    DDS_LifespanQosPolicy          lifespan;
    DDS_UserDataQosPolicy          user_data;
    DDS_OwnershipQosPolicy         ownership;
    DDS_OwnershipStrengthQosPolicy ownership_strength;
    DDS_WriterDataLifecycleQosPolicy writer_data_lifecycle; };
struct DDS_PublisherQos

```

```

        { DDS_PresentationQosPolicy      presentation;
          DDS_PartitionQosPolicy          partition;
          DDS_GroupDataQosPolicy          group_data;
          DDS_EntityFactoryQosPolicy      entity_factory; };
struct DDS_DataReaderQos
{
    DDS_DurabilityQosPolicy      durability;
    DDS_DeadlineQosPolicy        deadline;
    DDS_LatencyBudgetQosPolicy    latency_budget;
    DDS_LivelinessQosPolicy       liveliness;
    DDS_ReliabilityQosPolicy       reliability;
    DDS_DestinationOrderQosPolicy destination_order;
    DDS_HistoryQosPolicy           history;
    DDS_ResourceLimitsQosPolicy    resource_limits;
    DDS_UserDataQosPolicy          user_data;
    DDS_OwnershipQosPolicy         ownership;
    DDS_TimeBasedFilterQosPolicy    time_based_filter;
    DDS_ReaderDataLifecyleQosPolicy reader_data_lifecycle;};
struct DDS_SubscriberQos
{
    DDS_PresentationQosPolicy      presentation;
    DDS_PartitionQosPolicy          partition;
    DDS_GroupDataQosPolicy          group_data;
    DDS_EntityFactoryQosPolicy      entity_factory; };
/*
 * define <DDS_Entity>_QOS_DEFAULT
 */
#define DDS_PARTICIPANT_QOS_DEFAULT
#define DDS_TOPIC_QOS_DEFAULT
#define DDS_DATAWRITER_QOS_DEFAULT
#define DDS_PUBLISHER_QOS_DEFAULT
#define DDS_DATAREADER_QOS_DEFAULT
#define DDS_SUBSCRIBER_QOS_DEFAULT
#define DDS_DATAWRITER_QOS_USE_TOPIC_QOS
#define DDS_DATAREADER_QOS_USE_TOPIC_QOS
/*
 * implemented API operations
 *   <no operations>
 */

```

The next paragraphs describe the usage of each <DDS_Entity>Qos struct.

DDS_DataReaderQos

Synopsis

```

#include <dds_dcps.h>

struct DDS_DataReaderQos
{
    DDS_DurabilityQosPolicy      durability;
    DDS_DeadlineQosPolicy        deadline;
    DDS_LatencyBudgetQosPolicy    latency_budget;

```

```

DDS_LivelinessQosPolicy          liveliness;
DDS_ReliabilityQosPolicy         reliability;
DDS_DestinationOrderQosPolicy    destination_order;
DDS_HistoryQosPolicy             history;
DDS_ResourceLimitsQosPolicy      resource_limits;
DDS_UserDataQosPolicy            user_data;
DDS_OwnershipQosPolicy           ownership;
DDS_TimeBasedFilterQosPolicy     time_based_filter;
DDS_ReaderDataLifecycleQosPolicy reader_data_lifecycle;};

```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a DDS_DataReader.

Attributes

DDS_DurabilityQosPolicy durability - whether the data should be stored for late joining readers.

DDS_DeadlineQosPolicy deadline - the period within which a new sample is expected.

DDS_LatencyBudgetQosPolicy latency_budget - used by the Data Distribution Service for optimization.

DDS_LivelinessQosPolicy liveliness - the way the liveliness of the DDS_DataReader is asserted to the Data Distribution Service.

DDS_ReliabilityQosPolicy reliability - the reliability of the data distribution.

DDS_DestinationOrderQosPolicy destination_order - the order in which the DDS_DataReader timely orders the data.

DDS_HistoryQosPolicy history - how samples should be stored.

DDS_ResourceLimitsQosPolicy resource_limits - the maximum amount of resources to be used.

DDS_UserDataQosPolicy user_data - used to attach additional information to the DDS_DataReader.

DDS_OwnershipQosPolicy ownership - whether a DataWriter exclusively owns an instance.

DDS_TimeBasedFilterQosPolicy time_based_filter - the maximum data rate at which the DDS_DataReader will receive changes.

DDS_ReaderDataLifecycleQosPolicy reader_data_lifecycle - the minimum time after which a data instance is disposed of when the *instance_state* has become *DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE* or *DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE*.

Detailed Description

A *QosPolicy* can be set when the *DDS_DataReader* is created with the *DDS_Subscriber_create_datareader* operation (or modified with the *DDS_DataReader_set_qos* operation). Both operations take the *DDS_DataReaderQos* struct as a parameter. There may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified when they are being created and, in case they are already enabled, via the *DDS_DataReader_set_qos* operation.

Some *QosPolicy* have “immutable” semantics meaning that they can only be specified either at *DDS_DataReader* creation time or prior to calling the *DDS_DataReader_enable* operation on the *DDS_DataReader*.

See *Struct QosPolicy* on page 55 for a list of all <name>*QosPolicy* settings, their meaning, characteristics and possible values, as well as if it applies to a *DDS_DataReader*.

The initial value of the default *DDS_DataReaderQos* in the *DDS_Subscriber* are given in the following table:

Table 18 DDS_DATAREADER_QOS_DEFAULT

QosPolicy	Attribute	Value
durability	kind	DDS_VOLATILE_DURABILITY_QOS
deadline	period	DDS_DURATION_INFINITE
latency_budget	duration	0
liveliness	kind	DDS_AUTOMATIC_LIVELINESS_QOS
	lease_duration	DDS_DURATION_INFINITE
reliability	kind	DDS_BEST_EFFORT_RELIABILITY_QOS
	max_blocking_time	100 ms
	synchronous	FALSE
destination_order	kind	DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
history	kind	DDS_KEEP_LAST_HISTORY_QOS
	depth	1

Table 18 DDS_DATAREADER_QOS_DEFAULT

QosPolicy	Attribute	Value
resource_limits	max_samples	DDS_LENGTH_UNLIMITED
	max_instances	DDS_LENGTH_UNLIMITED
	max_samples_per_instance	DDS_LENGTH_UNLIMITED
user_data	value.length	0
ownership	kind	DDS_SHARED_OWNERSHIP_QOS
time_based_filter	minimum_separation	0
reader_data_lifecycle	autopurge_ nowriter_samples_delay	DDS_DURATION_INFINITE
	autopurge_ disposed_samples_delay	DDS_DURATION_INFINITE

DDS_DataWriterQos

Synopsis

```
#include <dds_dcps.h>
struct DDS_DataWriterQos
{
    DDS_DurabilityQosPolicy      durability;
    DDS_DeadlineQosPolicy       deadline;
    DDS_LatencyBudgetQosPolicy   latency_budget;
    DDS_LivelinessQosPolicy      liveliness;
    DDS_ReliabilityQosPolicy     reliability;
    DDS_DestinationOrderQosPolicy destination_order;
    DDS_HistoryQosPolicy         history;
    DDS_ResourceLimitsQosPolicy  resource_limits;
    DDS_TransportPriorityQosPolicy transport_priority;
    DDS_LifespanQosPolicy        lifespan;
    DDS_UserDataQosPolicy        user_data;
    DDS_OwnershipQosPolicy       ownership;
    DDS_OwnershipStrengthQosPolicy ownership_strength;
    DDS_WriterDataLifecycleQosPolicy writer_data_lifecycle;
};
```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a DDS_DataWriter.

Attributes

DDS_DurabilityQosPolicy *durability* - whether the data should be stored for late joining readers.

DDS_DeadlineQosPolicy *deadline* - the period within which a new sample is written.

DDS_LatencyBudgetQosPolicy latency_budget - used by the Data Distribution Service for optimization.

DDS_LivelinessQosPolicy liveliness - the way the liveliness of the DDS_DataWriter is asserted to the Data Distribution Service.

DDS_ReliabilityQosPolicy reliability - the reliability of the data distribution.

DDS_DestinationOrderQosPolicy destination_order - the order in which the DDS_DataReader timely orders the data.

DDS_HistoryQosPolicy history - how samples should be stored.

DDS_ResourceLimitsQosPolicy resource_limits - the maximum amount of resources to be used.

DDS_TransportPriorityQosPolicy transport_priority - a priority hint for the underlying transport layer.

DDS_LifespanQosPolicy lifespan - the maximum duration of validity of the data written by the DDS_DataWriter.

DDS_UserDataQosPolicy user_data - used to attach additional information to the DDS_DataWriter.

DDS_OwnershipQosPolicy ownership - whether a DataWriter exclusively owns an instance.

DDS_OwnershipStrengthQosPolicy ownership_strength - the strength to determine the ownership.

DDS_WriterDataLifecycleQosPolicy writer_data_lifecycle - whether unregistered instances are disposed of automatically or not.

Detailed Description

A QosPolicy can be set when the DDS_DataWriter is created with the DDS_Publisher_create_datawriter operation (or modified with the DDS_DataWriter_set_qos operation). Both operations take the DDS_DataWriterQos struct as a parameter. There may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified when they are being created and, in case they are already enabled, via the DDS_DataWriter_set_qos operation.

Some QosPolicy have “immutable” semantics meaning that they can only be specified either at DDS_DataWriter creation time or prior to calling the DDS_DataWriter_enable operation on the DDS_DataWriter.

The *Struct QosPolicy* provides the list of all <name>QosPolicy settings, their meaning, characteristics and possible values, as well as if it applies to a DDS_DataWriter.

The initial value of the default DDS_DataWriterQos in the DDS_Publisher are given in the following table:

Table 19 DDS_DATAWRITER_QOS_DEFAULT

QosPolicy	Attribute	Value
durability	kind	DDS_VOLATILE_DURABILITY_QOS
deadline	period	DDS_DURATION_INFINITE
latency_budget	duration	0
liveliness	kind	DDS_AUTOMATIC_LIVELINESS_QOS
	lease_duration	DDS_DURATION_INFINITE
reliability	kind	DDS_BEST_EFFORT_RELIABILITY_QOS
	max_blocking_time	100 ms
	synchronous	FALSE
destination_order	kind	DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
history	kind	DDS_KEEP_LAST_HISTORY_QOS
	depth	1
resource_limits	max_samples	DDS_LENGTH_UNLIMITED
	max_instances	DDS_LENGTH_UNLIMITED
	max_samples_per_instance	DDS_LENGTH_UNLIMITED
transport_priority	value	0
lifespan	duration	DDS_DURATION_INFINITE
user_data	value.length	0
ownership	kind	DDS_SHARED_OWNERSHIP_QOS
ownership_strength	value	0
writer_data_lifecycle	autodispose_unregistered_instances	TRUE

DDS_DomainParticipantFactoryQos

Synopsis

```
#include <dds_dcps.h>
struct DDS_DomainParticipantFactoryQos
{ DDS_EntityFactoryQosPolicy entity_factory; };
```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a DDS_DomainParticipantFactory.

Attributes

DDS_EntityFactoryQosPolicy entity_factory - whether a just created DomainParticipant should be enabled.

Detailed Description

The *QosPolicy* cannot be set at creation time, since the *DDS_DomainParticipantFactory* is a pre-existing object that can only be obtained with the *DDS_DomainParticipantFactory_get_instance* operation or its alias *DDS_TheParticipantFactory*. Therefore its *QosPolicy* is initialized to a default value according to *Table 20*:

Table 20 Default Values for DDS_DomainParticipantFactoryQos

QosPolicy	Attribute	Value
entity_factory	autoenable_created_entities	TRUE

After creation the *QosPolicy* can be modified with the *DDS_DomainParticipantFactory_set_qos* operation, which takes the *DDS_DomainParticipantFactoryQos* struct as a parameter.

DDS_DomainParticipantQos

Synopsis

```
#include <dds_dcps.h>
struct DDS_DomainParticipantQos
{
    DDS_UserDataQosPolicy user_data;
    DDS_EntityFactoryQosPolicy entity_factory;
    DDS_SchedulingQosPolicy watchdog_scheduling;
    DDS_SchedulingQosPolicy listener_scheduling; };
```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a *DDS_DomainParticipant*.

Attributes

DDS_UserDataQosPolicy user_data - used to attach additional information to the *DDS_DomainParticipant*.

DDS_EntityFactoryQosPolicy entity_factory - whether a just created *DDS_Entity* should be enabled.

DDS_SchedulingQosPolicy watchdog_scheduling - the scheduling parameters used to create the watchdog thread.

DDS_SchedulingQosPolicy listener_scheduling - the scheduling parameters used to create the listener thread.

Detailed Description

A `DDS_DomainParticipant` will spawn different threads for different purposes:

- A listener thread is spawned to perform the callbacks to all `DDS_Listener` objects attached to the various `DDS_Entities` contained in the `DDS_DomainParticipant`. The scheduling parameters for this thread can be specified in the `listener_scheduling` field of the `DDS_DomainParticipantQos`.
- A watchdog thread is spawned to report the the Liveliness of all `DDS_Entities` contained in the `DDS_DomainParticipant` whose `DDS_LivelinessQosPolicyKind` in their `DDS_LivelinessQosPolicy` is set to `DDS_AUTOMATIC_LIVELINESS_QOS`. The scheduling parameters for this thread can be specified in the `watchdog_scheduling` field of the `DDS_DomainParticipantQos`.

A `QosPolicy` can be set when the `DDS_DomainParticipant` is created with the `DDS_DomainParticipantFactory_create_participant` operation (or modified with the `DDS_DomainParticipant_set_qos` operation). Both operations take the `DDS_DomainParticipantQos` struct as a parameter. There may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified when they are being created and, in case they are already enabled, via the `DDS_DomainParticipant_set_qos` operation.

Some `QosPolicy` have “immutable” semantics meaning that they can only be specified either at `DDS_DomainParticipant` creation time or prior to calling the `DDS_DomainParticipant_enable` operation on the `DDS_DomainParticipant`.

The initial value of the default `DDS_DomainParticipantQos` in the `DDS_DomainParticipantFactory` are given in the following table:

Table 21 DDS_PARTICIPANT_QOS_DEFAULT

QosPolicy	Attribute	Value
user_data	value.length	0
entity_factory	autoenable_created_entities	TRUE
watchdog_scheduling	scheduling_class.kind	SCHEDULE_DEFAULT
	scheduling_priority_kind.kind	PRIORITY_RELATIVE
	scheduling_priority	0

Table 21 DDS_PARTICIPANT_QOS_DEFAULT

QosPolicy	Attribute	Value
listener_scheduling	scheduling_class.kind	SCHEDULE_DEFAULT
	scheduling_priority_kind.kind	PRIORITY_RELATIVE
	scheduling_priority	0

DDS_PublisherQos

Synopsis

```
#include <dds_dcps.h>
struct DDS_PublisherQos
{
    DDS_PresentationQosPolicy    presentation;
    DDS_PartitionQosPolicy       partition;
    DDS_GroupDataQosPolicy       group_data;
    DDS_EntityFactoryQosPolicy   entity_factory; };
```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a DDS_Publisher.

Attributes

DDS_PresentationQosPolicy presentation - the dependency of changes to data-instances.

DDS_PartitionQosPolicy partition - the partitions in which the DDS_Publisher is active.

DDS_GroupDataQosPolicy group_data - used to attach additional information to the DDS_Publisher.

DDS_EntityFactoryQosPolicy entity_factory - whether a just created DDS_DataWriter should be enabled.

Detailed Description

A QosPolicy can be set when the DDS_Publisher is created with the DDS_DomainParticipant_create_publisher operation (or modified with the DDS_Publisher_set_qos operation). Both operations take the DDS_PublisherQos struct as a parameter. There may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified when they are being created and, in case they are already enabled, via the DDS_Publisher_set_qos operation.

Some `QosPolicy` have “immutable” semantics meaning that they can only be specified either at `DDS_Publisher` creation time or prior to calling the `DDS_Publisher_enable` operation on the `DDS_Publisher`.

The initial value of the default `DDS_PublisherQos` in the `DDS_DomainParticipant` are given in the following table:

Table 22 DDS_PUBLISHER_QOS_DEFAULT

QosPolicy	Attribute	Value
presentation	access_scope	DDS_INSTANCE_PRESENTATION_QOS
	coherent_access	FALSE
	ordered_access	FALSE
partition	name.length	0
group_data	value.length	0
entity_factory	autoenable_created_entities	TRUE

DDS_SubscriberQos

Synopsis

```
#include <dds_dcps.h>
struct DDS_SubscriberQos
{
    DDS_PresentationQosPolicy    presentation;
    DDS_PartitionQosPolicy      partition;
    DDS_GroupDataQosPolicy      group_data;
    DDS_EntityFactoryQosPolicy  entity_factory; };
```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a `DDS_Subscriber`.

Attributes

DDS_PresentationQosPolicy presentation - the dependency of changes to data-instances.

DDS_PartitionQosPolicy partition - the partitions in which the `DDS_Subscriber` is active.

DDS_GroupDataQosPolicy group_data - used to attach additional information to the `DDS_Subscriber`.

DDS_EntityFactoryQosPolicy entity_factory - whether a just created `DDS_DataReader` should be enabled.

Detailed Description

A `QosPolicy` can be set when the `DDS_Subscriber` is created with the `DDS_DomainParticipant_create_subscriber` operation (or modified with the `DDS_Subscriber_set_qos` operation). Both operations take the `DDS_SubscriberQos` struct as a parameter. There may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified when they are being created and, in case they are already enabled, via the `DDS_Subscriber_set_qos` operation.

Some `QosPolicy` have “immutable” semantics meaning that they can only be specified either at `DDS_Subscriber` creation time or prior to calling the `DDS_Subscriber_enable` operation on the `DDS_Subscriber`.

The initial value of the default `DDS_SubscriberQos` in the `DDS_DomainParticipant` are given in the following table:

Table 23 DDS_SUBSCRIBER_QOS_DEFAULT

QosPolicy	Attribute	Value
presentation	access_scope	DDS_INSTANCE_PRESENTATION_QOS
	coherent_access	FALSE
	ordered_access	FALSE
partition	name.length	0
group_data	value.length	0
entity_factory	autoenable_ created_entities	TRUE

DDS_TopicQos

Synopsis

```
#include <dds_dcps.h>
struct DDS_TopicQos
{
    DDS_TopicDataQosPolicy      topic_data;
    DDS_DurabilityQosPolicy     durability;
    DDS_DurabilityServiceQosPolicy durability_service;
    DDS_DeadlineQosPolicy       deadline;
    DDS_LatencyBudgetQosPolicy  latency_budget;
    DDS_LivelinessQosPolicy     liveliness;
    DDS_ReliabilityQosPolicy     reliability;
    DDS_DestinationOrderQosPolicy destination_order;
    DDS_HistoryQosPolicy        history;
    DDS_ResourceLimitsQosPolicy resource_limits;
    DDS_TransportPriorityQosPolicy transport_priority;
    DDS_LifespanQosPolicy       lifespan;
```

```
DDS_OwnershipQosPolicy ownership; };
```

Description

This struct provides the basic mechanism for an application to specify Quality of Service attributes for a DDS_Topic.

Attributes

DDS_TopicDataQosPolicy topic_data - used to attach additional information to the DDS_Topic.

DDS_DurabilityQosPolicy durability - whether the data should be stored for late joining readers.

DDS_DurabilityServiceQosPolicy durability_service - the behaviour of the “transient/persistent service” of the Data Distribution System regarding Transient and Persistent DDS_Topic instances.

DDS_DeadlineQosPolicy deadline - the period within which a new sample is expected or written.

DDS_LatencyBudgetQosPolicy latency_budget - used by the Data Distribution Service for optimization.

DDS_LivelinessQosPolicy liveliness - the way the liveliness of the DDS_Topic is asserted to the Data Distribution Service.

DDS_ReliabilityQosPolicy reliability - the reliability of the data distribution.

DDS_DestinationOrderQosPolicy destination_order - the order in which the DDS_DataReader timely orders the data.

DDS_HistoryQosPolicy history - how samples should be stored.

DDS_ResourceLimitsQosPolicy resource_limits - the maximum amount of resources to be used.

DDS_TransportPriorityQosPolicy transport_priority - a priority hint for the underlying transport layer.

DDS_LifespanQosPolicy lifespan - the maximum duration of validity of the data written by a DDS_DataWriter.

DDS_OwnershipQosPolicy ownership - whether a DDS_DataWriter exclusively owns an instance.

Detailed Description

A QosPolicy can be set when the DDS_Topic is created with the DDS_DomainParticipant_create_topic operation (or modified with the DDS_Topic_set_qos operation). Both operations take the DDS_TopicQos struct

as a parameter. There may be cases where several policies are in conflict. Consistency checking is performed each time the policies are modified when they are being created and, in case they are already enabled, via the `DDS_Topic_set_qos` operation.

Some `QosPolicy` have “immutable” semantics meaning that they can only be specified either at `DDS_Topic` creation time or prior to calling the `DDS_Topic_enable` operation on the `DDS_Topic`.

The initial value of the default `DDS_TopicQos` in the `DDS_DomainParticipant` are given in the following table:

Table 24 DDS_TOPIC_QOS_DEFAULT

QosPolicy	Attribute	Value
topic_data	value.length	0
durability	kind	DDS_VOLATILE_DURABILITY_QOS
durability_service	service_cleanup_delay	0
	history_kind	DDS_KEEP_LAST_HISTORY_QOS
	history_depth	1
	max_samples	DDS_LENGTH_UNLIMITED
	max_instances	DDS_LENGTH_UNLIMITED
	max_samples_per_instance	DDS_LENGTH_UNLIMITED
deadline	period	DDS_DURATION_INFINITE
latency_budget	duration	0
liveliness	kind	DDS_AUTOMATIC_LIVELINESS_QOS
	lease_duration	DDS_DURATION_INFINITE
reliability	kind	DDS_BEST_EFFORT_RELIABILITY_QOS
	max_blocking_time	100 ms
	synchronous	FALSE
destination_order	kind	DDS_BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS
history	kind	DDS_KEEP_LAST_HISTORY_QOS
	depth	1
resource_limits	max_samples	DDS_LENGTH_UNLIMITED
	max_instances	DDS_LENGTH_UNLIMITED
	max_samples_per_instance	DDS_LENGTH_UNLIMITED

Table 24 DDS_TOPIC_QOS_DEFAULT (Continued)

QosPolicy	Attribute	Value
transport_priority	value	0
lifespan	duration	DDS_DURATION_INFINITE
ownership	kind	DDS_SHARED_OWNERSHIP_QOS

Appendix

B *API Constants and Types*

These constants and types are taken from the `dds_dcps.h` include file.

```
/* Duration and Time
*/
struct DDS_Duration_t
{
    DDS_long sec;
    DDS_unsigned_long nanosec;
};

#define DDS_DURATION_INFINITE_SEC          0x7fffffff
#define DDS_DURATION_INFINITE_NSEC        0x7fffffffU
#define DDS_DURATION_ZERO_SEC              0
#define DDS_DURATION_ZERO_NSEC             0U
#define DDS_DURATION_INFINITE              {
    DDS_DURATION_INFINITE_SEC,
    DDS_DURATION_INFINITE_NSEC }
#define DDS_DURATION_ZERO                  {
    DDS_DURATION_ZERO_SEC,
    DDS_DURATION_ZERO_NSEC }

struct DDS_Time_t
{
    DDS_long sec;
    DDS_unsigned_long nanosec;
};

/*
 * Pre-defined values
 */
#define DDS_HANDLE_NIL                     DDS_HANDLE_NIL_NATIVE
#define DDS_LENGTH_UNLIMITED               -1
#define DDS_TIMESTAMP_INVALID_SEC           -1
#define DDS_TIMESTAMP_INVALID_NSEC         4294967295U
#define DDS_TIMESTAMP_INVALID              {
    DDS_TIMESTAMP_INVALID_SEC,
    DDS_TIMESTAMP_INVALID_NSEC }

/* -----
 * Return codes
 * ----- */
#define DDS_RETCODE_OK                      0
```

```

#define DDS_RETCODE_ERROR 1
#define DDS_RETCODE_UNSUPPORTED 2
#define DDS_RETCODE_BAD_PARAMETER 3
#define DDS_RETCODE_PRECONDITION_NOT_MET 4
#define DDS_RETCODE_OUT_OF_RESOURCES 5
#define DDS_RETCODE_NOT_ENABLED 6
#define DDS_RETCODE_IMMUTABLE_POLICY 7
#define DDS_RETCODE_INCONSISTENT_POLICY 8
#define DDS_RETCODE_ALREADY_DELETED 9
#define DDS_RETCODE_TIMEOUT 10
#define DDS_RETCODE_NO_DATA 11
#define DDS_RETCODE_ILLEGAL_OPERATION 12

/* -----
 * DDS_Status to support listeners and conditions
 * ----- */
#define DDS_INCONSISTENT_TOPIC_STATUS 1U
#define DDS_OFFERED_DEADLINE_MISSED_STATUS 2U
#define DDS_REQUESTED_DEADLINE_MISSED_STATUS 4U
#define DDS_OFFERED_INCOMPATIBLE_QOS_STATUS 32U
#define DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS 64U
#define DDS_SAMPLE_LOST_STATUS 128U
#define DDS_SAMPLE_REJECTED_STATUS 256U
#define DDS_DATA_ON_READERS_STATUS 512U
#define DDS_DATA_AVAILABLE_STATUS 1024U
#define DDS_LIVELINESS_LOST_STATUS 2048U
#define DDS_LIVELINESS_CHANGED_STATUS 4096U
#define DDS_PUBLICATION_MATCHED_STATUS 8192U
#define DDS_SUBSCRIPTION_MATCHED_STATUS 16384U

#define DDS_ANY_STATUS 0x7FE7
#define DDS_STATUS_MASK_ANY_V1_2 0x7FE7
#define DDS_STATUS_MASK_NONE 0x0

/*
 * States
 * */
/*
 * Sample states to support reads
 */
#define DDS_READ_SAMPLE_STATE 1U
#define DDS_NOT_READ_SAMPLE_STATE 2U
/*
 * This is a bit-mask DDS_SampleStateKind
 */
#define DDS_ANY_SAMPLE_STATE 65535U

/*

```

```

* View states to support reads
*/
#define DDS_NEW_VIEW_STATE 1U
#define DDS_NOT_NEW_VIEW_STATE 2U
/*
* This is a bit-mask DDS_ViewStateKind
*/
#define DDS_ANY_VIEW_STATE 65535U

/*
* Instance states to support reads
*/
#define DDS_ALIVE_INSTANCE_STATE 1U
#define DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE 2U
#define DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE 4U

/*
* This is a bit-mask DDS_InstanceStateKind
*/
#define DDS_ANY_INSTANCE_STATE 65535U
#define DDS_NOT_ALIVE_INSTANCE_STATE 6U

/*
* Participant Factory define
*/
#define TheParticipantFactory
        (DDS_DomainParticipantFactory_get_instance())

/*
* Qos defines
* */
#define DDS_PARTICIPANT_QOS_DEFAULT NULL
#define DDS_TOPIC_QOS_DEFAULT NULL
#define DDS_PUBLISHER_QOS_DEFAULT NULL
#define DDS_SUBSCRIBER_QOS_DEFAULT NULL
#define DDS_DATAREADER_QOS_DEFAULT NULL
#define DDS_DATAWRITER_QOS_DEFAULT NULL
#define DDS_DATAWRITER_QOS_USE_TOPIC_QOS ((DDS_DataWriterQos *)-1)
#define DDS_DATAREADER_QOS_USE_TOPIC_QOS ((DDS_DataReaderQos *)-1)

/* QosPolicy
* /
#define DDS_USERDATA_QOS_POLICY_NAME "UserData"
#define DDS_DURABILITY_QOS_POLICY_NAME "Durability"
#define DDS_PRESENTATION_QOS_POLICY_NAME "Presentation"
#define DDS_DEADLINE_QOS_POLICY_NAME "Deadline"
#define DDS_LATENCYBUDGET_QOS_POLICY_NAME "LatencyBudget"
#define DDS_OWNERSHIP_QOS_POLICY_NAME "Ownership"
#define DDS_OWNERSHIPSTRENGTH_QOS_POLICY_NAME "OwnershipStrength"

```

```

#define DDS_LIVELINESS_QOS_POLICY_NAME          "Liveliness"
#define DDS_TIMEBASEDFILTER_QOS_POLICY_NAME      "TimeBasedFilter"
#define DDS_PARTITION_QOS_POLICY_NAME            "Partition"
#define DDS_RELIABILITY_QOS_POLICY_NAME          "Reliability"
#define DDS_DESTINATIONORDER_QOS_POLICY_NAME     "DestinationOrder"
#define DDS_HISTORY_QOS_POLICY_NAME              "History"
#define DDS_RESOURCELIMITS_QOS_POLICY_NAME        "ResourceLimits"
#define DDS_ENTITYFACTORY_QOS_POLICY_NAME        "EntityFactory"
#define DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_NAME
    "WriterDataLifecycle"
#define DDS_READERDATA_LIFECYCLE_QOS_POLICY_NAME
    "ReaderDataLifecycle"
#define DDS_TOPICDATA_QOS_POLICY_NAME            "TopicData"
#define DDS_GROUPDATA_QOS_POLICY_NAME            "GroupData"
#define DDS_TRANSPORTPRIORITY_QOS_POLICY_NAME    "TransportPriority"
#define DDS_LIFESPAN_QOS_POLICY_NAME             "Lifespan"
#define DDS_DURABILITYSERVICE_QOS_POLICY_NAME   "DurabilityService"

#define DDS_INVALID_QOS_POLICY_ID                0
#define DDS_USERDATA_QOS_POLICY_ID               1
#define DDS_DURABILITY_QOS_POLICY_ID              2
#define DDS_PRESENTATION_QOS_POLICY_ID            3
#define DDS_DEADLINE_QOS_POLICY_ID                4
#define DDS_LATENCYBUDGET_QOS_POLICY_ID           5
#define DDS_OWNERSHIP_QOS_POLICY_ID               6
#define DDS_OWNERSHIPSTRENGTH_QOS_POLICY_ID       7
#define DDS_LIVELINESS_QOS_POLICY_ID              8
#define DDS_TIMEBASEDFILTER_QOS_POLICY_ID         9
#define DDS_PARTITION_QOS_POLICY_ID               10
#define DDS_RELIABILITY_QOS_POLICY_ID              11
#define DDS_DESTINATIONORDER_QOS_POLICY_ID         12
#define DDS_HISTORY_QOS_POLICY_ID                  13
#define DDS_RESOURCELIMITS_QOS_POLICY_ID           14
#define DDS_ENTITYFACTORY_QOS_POLICY_ID            15
#define DDS_WRITERDATA_LIFECYCLE_QOS_POLICY_ID     16
#define DDS_READERDATA_LIFECYCLE_QOS_POLICY_ID     17
#define DDS_TOPICDATA_QOS_POLICY_ID               18
#define DDS_GROUPDATA_QOS_POLICY_ID                19
#define DDS_TRANSPORTPRIORITY_QOS_POLICY_ID        20
#define DDS_LIFESPAN_QOS_POLICY_ID                 21
#define DDS_DURABILITYSERVICE_QOS_POLICY_ID       22

```



Platform Specific IDL Interface

The IDL code in the next paragraphs are taken from the *OMG C Language Mapping Specification*.

dds_dcps.idl

```
#define DOMAINID_TYPE_NATIVE string
#define HANDLE_TYPE_NATIVE long long
#define HANDLE_NIL_NATIVE 0
#define BUILTIN_TOPIC_KEY_TYPE_NATIVE long long
#define TheParticipantFactory
#define PARTICIPANT_QOS_DEFAULT
#define TOPIC_QOS_DEFAULT
#define PUBLISHER_QOS_DEFAULT
#define SUBSCRIBER_QOS_DEFAULT
#define DATAWRITER_QOS_DEFAULT
#define DATAREADER_QOS_DEFAULT
#define DATAWRITER_QOS_USE_TOPIC_QOS
#define DATAREADER_QOS_USE_TOPIC_QOS
module DDS {
    typedef DOMAINID_TYPE_NATIVE DomainId_t;
    typedef HANDLE_TYPE_NATIVE InstanceHandle_t;
    typedef BUILTIN_TOPIC_KEY_TYPE_NATIVE BuiltinTopicKey_t[3];
    typedef sequence<InstanceHandle_t> InstanceHandleSeq;
    typedef long ReturnCode_t;
    typedef long QosPolicyId_t;
    typedef sequence<string> StringSeq;
    struct Duration_t {
        long sec;
        unsigned long nanosec;
    };
    struct Time_t {
        long sec;
        unsigned long nanosec;
    };
    //
    // Pre-defined values
    //
    const InstanceHandle_t HANDLE_NIL= HANDLE_NIL_NATIVE;
    const long LENGTH_UNLIMITED= -1;
    const long DURATION_INFINITE_SEC= 0x7fffffff;
    const unsigned long DURATION_INFINITE_NSEC= 0x7fffffff;
    const long DURATION_ZERO_SEC= 0;
```

```

const unsigned long DURATION_ZERO_NSEC= 0;
const long TIMESTAMP_INVALID_SEC= -1;
const unsigned long TIMESTAMP_INVALID_NSEC= 0xffffffff;
//
// Return codes
//
const ReturnCode_t RETCODE_OK = 0;
const ReturnCode_t RETCODE_ERROR = 1;
const ReturnCode_t RETCODE_UNSUPPORTED = 2;
const ReturnCode_t RETCODE_BAD_PARAMETER = 3;
const ReturnCode_t RETCODE_PRECONDITION_NOT_MET = 4;
const ReturnCode_t RETCODE_OUT_OF_RESOURCES = 5;
const ReturnCode_t RETCODE_NOT_ENABLED = 6;
const ReturnCode_t RETCODE_IMMUTABLE_POLICY = 7;
const ReturnCode_t RETCODE_INCONSISTENT_POLICY = 8;
const ReturnCode_t RETCODE_ALREADY_DELETED = 9;
const ReturnCode_t RETCODE_TIMEOUT = 10;
const ReturnCode_t RETCODE_NO_DATA = 11;
const ReturnCode_t RETCODE_ILLEGAL_OPERATION = 12;

//
// Status to support listeners and conditions
//
typedef unsigned long StatusKind;
typedef unsigned long StatusMask; // bit-mask StatusKind
const StatusKind INCONSISTENT_TOPIC_STATUS = 0x0001 << 0;
const StatusKind OFFERED_DEADLINE_MISSED_STATUS = 0x0001 << 1;
const StatusKind REQUESTED_DEADLINE_MISSED_STATUS = 0x0001 << 2;
const StatusKind OFFERED_INCOMPATIBLE_QOS_STATUS = 0x0001 << 5;
const StatusKind REQUESTED_INCOMPATIBLE_QOS_STATUS= 0x0001 << 6;
const StatusKind SAMPLE_LOST_STATUS = 0x0001 << 7;
const StatusKind SAMPLE_REJECTED_STATUS = 0x0001 << 8;
const StatusKind DATA_ON_READERS_STATUS = 0x0001 << 9;
const StatusKind DATA_AVAILABLE_STATUS = 0x0001 << 10;
const StatusKind LIVELINESS_LOST_STATUS = 0x0001 << 11;
const StatusKind LIVELINESS_CHANGED_STATUS = 0x0001 << 12;
const StatusKind PUBLICATION_MATCHED_STATUS = 0x0001 << 13;
const StatusKind SUBSCRIPTION_MATCHED_STATUS = 0x0001 << 14;
struct InconsistentTopicStatus {
    long total_count;
    long total_count_change;
};
struct SampleLostStatus {
    long total_count;
    long total_count_change;
};
enum SampleRejectedStatusKind {
    NOT_REJECTED,
    REJECTED_BY_INSTANCE_LIMIT,
    REJECTED_BY_SAMPLES_LIMIT,

```



```

    REJECTED_BY_SAMPLES_PER_INSTANCE_LIMIT
};
struct SampleRejectedStatus {
    long total_count;
    long total_count_change;
    SampleRejectedStatusKind last_reason;
    InstanceHandle_t last_instance_handle;
};
struct LivelinessLostStatus {
    long total_count;
    long total_count_change;
};
struct LivelinessChangedStatus {
    long alive_count;
    long not_alive_count;
    long alive_count_change;
    long not_alive_count_change;
    InstanceHandle_t last_publication_handle;
};
struct OfferedDeadlineMissedStatus {
    long total_count;
    long total_count_change;
    InstanceHandle_t last_instance_handle;
};
struct RequestedDeadlineMissedStatus {
    long total_count;
    long total_count_change;
    InstanceHandle_t last_instance_handle;
};
struct QosPolicyCount {
    QosPolicyId_t policy_id;
    long count;
};
typedef sequence<QosPolicyCount> QosPolicyCountSeq;
struct OfferedIncompatibleQosStatus {
    long total_count;
    long total_count_change;
    QosPolicyId_t last_policy_id;
    QosPolicyCountSeq policies;
};
struct RequestedIncompatibleQosStatus {
    long total_count;
    long total_count_change;
    QosPolicyId_t last_policy_id;
    QosPolicyCountSeq policies;
};
struct PublicationMatchedStatus {
    long total_count;
    long total_count_change;
    long current_count;
};

```

```

        long current_count_change;
        InstanceHandle_t last_subscription_handle;
};
struct SubscriptionMatchedStatus {
    long total_count;
    long total_count_change;
    long current_count;
    long current_count_change;
    InstanceHandle_t last_publication_handle;
};
//
// Listeners
//
interface Listener;
interface Entity;
interface TopicDescription;
interface Topic;
interface ContentFilteredTopic;
interface MultiTopic;
interface DataWriter;
interface DataReader;
interface Subscriber;
interface Publisher;
typedef sequence<Topic> TopicSeq;
typedef sequence<DataReader> DataReaderSeq;
interface Listener {
};
interface TopicListener : Listener {
void
    on_inconsistent_topic(
        in Topic the_topic,
        in InconsistentTopicStatus status);
};
interface DataWriterListener : Listener {
void
    on_offered_deadline_missed(
        in DataWriter writer,
        in OfferedDeadlineMissedStatus status);
void
    on_offered_incompatible_qos(
        in DataWriter writer,
        in OfferedIncompatibleQosStatus status);
void
    on_liveliness_lost(
        in DataWriter writer,
        in LivelinessLostStatus status);
void
    on_publication_matched(
        in DataWriter writer,
        in PublicationMatchedStatus status);

```

```

};
interface PublisherListener : DataWriterListener {
};
interface DataReaderListener : Listener {
void
    on_requested_deadline_missed(
        in DataReader reader,
        in RequestedDeadlineMissedStatus status);
void
    on_requested_incompatible_qos(
        in DataReader reader,
        in RequestedIncompatibleQosStatus status);
void
    on_sample_rejected(
        in DataReader reader,
        in SampleRejectedStatus status);
void
    on_liveliness_changed(
        in DataReader reader,
        in LivelinessChangedStatus status);
void
    on_data_available(
        in DataReader reader);
void
    on_subscription_matched(
        in DataReader reader,
        in SubscriptionMatchedStatus status);
void
    on_sample_lost(
        in DataReader reader,
        in SampleLostStatus status);
};
interface SubscriberListener : DataReaderListener {
void
    on_data_on_readers(
        in Subscriber subs);
};
interface DomainParticipantListener : TopicListener,
                                     PublisherListener,
                                     SubscriberListener {
};
//
// Conditions
//
interface Condition {
boolean
get_trigger_value();
};
typedef sequence<Condition> ConditionSeq;
interface WaitSet {

```

```

ReturnCode_t
wait(
    inout ConditionSeq active_conditions,
    in Duration_t timeout);
ReturnCode_t
attach_condition(
    in Condition cond);
ReturnCode_t
detach_condition(
    in Condition cond);
ReturnCode_t
get_conditions(
    inout ConditionSeq attached_conditions);
};
interface GuardCondition : Condition {
ReturnCode_t
set_trigger_value(
    in boolean value);
};
interface StatusCondition : Condition {
StatusMask
get_enabled_statuses();
ReturnCode_t
set_enabled_statuses(
    in StatusMask mask);
Entity
get_entity();
};
// Sample states to support reads
typedef unsigned long SampleStateKind;
typedef sequence <SampleStateKind> SampleStateSeq;
const SampleStateKind READ_SAMPLE_STATE = 0x0001 << 0;
const SampleStateKind NOT_READ_SAMPLE_STATE = 0x0001 << 1;
// This is a bit-mask SampleStateKind
typedef unsigned long SampleStateMask;
const SampleStateMask ANY_SAMPLE_STATE = 0xffff;
// View states to support reads
typedef unsigned long ViewStateKind;
typedef sequence<ViewStateKind> ViewStateSeq;
const ViewStateKind NEW_VIEW_STATE = 0x0001 << 0;
const ViewStateKind NOT_NEW_VIEW_STATE = 0x0001 << 1;
// This is a bit-mask ViewStateKind
typedef unsigned long ViewStateMask;
const ViewStateMask ANY_VIEW_STATE = 0xffff;
// Instance states to support reads
typedef unsigned long InstanceStateKind;
typedef sequence<InstanceStateKind> InstanceStateSeq;
const InstanceStateKind ALIVE_INSTANCE_STATE = 0x0001 << 0;
const InstanceStateKind NOT_ALIVE_DISPOSED_INSTANCE_STATE = 0x0001
    << 1;

```

```

const InstanceStateKind NOT_ALIVE_NO_WRITERS_INSTANCE_STATE =
    0x0001 << 2;
// This is a bit-mask InstanceStateKind
typedef unsigned long InstanceStateMask;
const InstanceStateMask ANY_INSTANCE_STATE = 0xffff;
const InstanceStateMask NOT_ALIVE_INSTANCE_STATE = 0x006;
interface ReadCondition : Condition {
SampleStateMask
get_sample_state_mask();
ViewStateMask
get_view_state_mask();
InstanceStateMask
get_instance_state_mask();
DataReader
get_datareader();
};
interface QueryCondition : ReadCondition {
string
get_query_expression();
ReturnCode_t
get_query_parameters(
    inout StringSeq query_parameters);
ReturnCode_t
set_query_parameters(
    in StringSeq query_parameters);
};
//
// Qos
//
const string USERDATA_QOS_POLICY_NAME      = "UserData";
const string DURABILITY_QOS_POLICY_NAME    = "Durability";
const string PRESENTATION_QOS_POLICY_NAME  = "Presentation";
const string DEADLINE_QOS_POLICY_NAME      = "Deadline";
const string LATENCYBUDGET_QOS_POLICY_NAME = "LatencyBudget";
const string OWNERSHIP_QOS_POLICY_NAME     = "Ownership";
const string OWNERSHIPSTRENGTH_QOS_POLICY_NAME=
    "OwnershipStrength";
const string LIVELINESS_QOS_POLICY_NAME    = "Liveliness";
const string TIMEBASEDFILTER_QOS_POLICY_NAME= "TimeBasedFilter";
const string PARTITION_QOS_POLICY_NAME     = "Partition";
const string RELIABILITY_QOS_POLICY_NAME   = "Reliability";
const string DESTINATIONORDER_QOS_POLICY_NAME =
    "DestinationOrder";
const string HISTORY_QOS_POLICY_NAME       = "History";
const string RESOURCELIMITS_QOS_POLICY_NAME= "ResourceLimits";
const string ENTITYFACTORY_QOS_POLICY_NAME = "EntityFactory";
const string WRITERDATA LifecYCLE_QOS_POLICY_NAME=
    "WriterDataLifecYCLE";
const string READERDATA LifecYCLE_QOS_POLICY_NAME=
    "ReaderDataLifecYCLE";

```

```

const string TOPICDATA_QOS_POLICY_NAME          = "TopicData";
const string GROUPDATA_QOS_POLICY_NAME          = "GroupData";
const string TRANSPORTPRIORITY_QOS_POLICY_NAME=
    "TransportPriority";
const string LIFESPAN_QOS_POLICY_NAME          = "Lifespan";
const string DURABILITYSERVICE_QOS_POLICY_NAME=
    "DurabilityService";
const QosPolicyId_t INVALID_QOS_POLICY_ID      = 0;
const QosPolicyId_t USERDATA_QOS_POLICY_ID    = 1;
const QosPolicyId_t DURABILITY_QOS_POLICY_ID   = 2;
const QosPolicyId_t PRESENTATION_QOS_POLICY_ID = 3;
const QosPolicyId_t DEADLINE_QOS_POLICY_ID    = 4;
const QosPolicyId_t LATENCYBUDGET_QOS_POLICY_ID = 5;
const QosPolicyId_t OWNERSHIP_QOS_POLICY_ID    = 6;
const QosPolicyId_t OWNERSHIPSTRENGTH_QOS_POLICY_ID = 7;
const QosPolicyId_t LIVELINESS_QOS_POLICY_ID   = 8;
const QosPolicyId_t TIMEBASEDFILTER_QOS_POLICY_ID = 9;
const QosPolicyId_t PARTITION_QOS_POLICY_ID    = 10;
const QosPolicyId_t RELIABILITY_QOS_POLICY_ID  = 11;
const QosPolicyId_t DESTINATIONORDER_QOS_POLICY_ID = 12;
const QosPolicyId_t HISTORY_QOS_POLICY_ID      = 13;
const QosPolicyId_t RESOURCELIMITS_QOS_POLICY_ID = 14;
const QosPolicyId_t ENTITYFACTORY_QOS_POLICY_ID = 15;
const QosPolicyId_t WRITERDATALIFECYCLE_QOS_POLICY_ID= 16;
const QosPolicyId_t READERDATALIFECYCLE_QOS_POLICY_ID= 17;
const QosPolicyId_t TOPICDATA_QOS_POLICY_ID   = 18;
const QosPolicyId_t GROUPDATA_QOS_POLICY_ID   = 19;
const QosPolicyId_t TRANSPORTPRIORITY_QOS_POLICY_ID = 20;
const QosPolicyId_t LIFESPAN_QOS_POLICY_ID    = 21;
const QosPolicyId_t DURABILITYSERVICE_QOS_POLICY_ID = 22;
struct UserDataQosPolicy {
sequence<octet> value;
};
struct TopicDataQosPolicy {
sequence<octet> value;
};
struct GroupDataQosPolicy {
sequence<octet> value;
};
struct TransportPriorityQosPolicy {
long value;
};
struct LifespanQosPolicy {
Duration_t duration;
};
enum DurabilityQosPolicyKind {
VOLATILE_DURABILITY_QOS,
TRANSIENT_LOCAL_DURABILITY_QOS,
TRANSIENT_DURABILITY_QOS,
PERSISTENT_DURABILITY_QOS

```

```

};
struct DurabilityQosPolicy {
DurabilityQosPolicyKind kind;
};
enum PresentationQosPolicyAccessScopeKind {
INSTANCE_PRESENTATION_QOS,
TOPIC_PRESENTATION_QOS,
GROUP_PRESENTATION_QOS
};
struct PresentationQosPolicy {
    PresentationQosPolicyAccessScopeKind access_scope;
    boolean coherent_access;
    boolean ordered_access;
};
struct DeadlineQosPolicy {
    Duration_t period;
};
struct LatencyBudgetQosPolicy {
    Duration_t duration;
};
enum OwnershipQosPolicyKind {
    SHARED_OWNERSHIP_QOS,
    EXCLUSIVE_OWNERSHIP_QOS
};
struct OwnershipQosPolicy {
    OwnershipQosPolicyKind kind;
};
struct OwnershipStrengthQosPolicy {
    long value;
};
enum LivelinessQosPolicyKind {
    AUTOMATIC_LIVELINESS_QOS,
    MANUAL_BY_PARTICIPANT_LIVELINESS_QOS,
    MANUAL_BY_TOPIC_LIVELINESS_QOS
};
struct LivelinessQosPolicy {
    LivelinessQosPolicyKind kind;
    Duration_t lease_duration;
};
struct TimeBasedFilterQosPolicy {
    Duration_t minimum_separation;
};
struct PartitionQosPolicy {
    StringSeq name;
};
enum ReliabilityQosPolicyKind {
    BEST_EFFORT_RELIABILITY_QOS,
    RELIABLE_RELIABILITY_QOS
};
struct ReliabilityQosPolicy {

```

```

        ReliabilityQosPolicyKind kind;
        Duration_t max_blocking_time;
        boolean synchronous;
    };
    enum DestinationOrderQosPolicyKind {
        BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS,
        BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS
    };
    struct DestinationOrderQosPolicy {
        DestinationOrderQosPolicyKind kind;
    };
    enum HistoryQosPolicyKind {
        KEEP_LAST_HISTORY_QOS,
        KEEP_ALL_HISTORY_QOS
    };
    struct HistoryQosPolicy {
        HistoryQosPolicyKind kind;
        long depth;
    };
    struct ResourceLimitsQosPolicy {
        long max_samples;
        long max_instances;
        long max_samples_per_instance;
    };
    struct EntityFactoryQosPolicy {
        boolean autoenable_created_entities;
    };
    struct WriterDataLifecycleQosPolicy {
        boolean autodispose_unregistered_instances;
    };
    struct ReaderDataLifecycleQosPolicy {
        Duration_t autopurge_nowriter_samples_delay;
        Duration_t autopurge_disposed_samples_delay;
    };
    struct DurabilityServiceQosPolicy {
        Duration_t service_cleanup_delay;
        HistoryQosPolicyKind history_kind;
        long history_depth;
        long max_samples;
        long max_instances;
        long max_samples_per_instance;
    };
    struct DomainParticipantFactoryQos {
        EntityFactoryQosPolicy entity_factory;
    };
    struct DomainParticipantQos {
        UserDataQosPolicy user_data;
        EntityFactoryQosPolicy entity_factory;
    };
    struct TopicQos {

```



```

TopicDataQosPolicy topic_data;
DurabilityQosPolicy durability;
DurabilityServiceQosPolicy durability_service;
DeadlineQosPolicy deadline;
LatencyBudgetQosPolicy latency_budget;
LivelinessQosPolicy liveliness;
ReliabilityQosPolicy reliability;
DestinationOrderQosPolicy destination_order;
HistoryQosPolicy history;
ResourceLimitsQosPolicy resource_limits;
TransportPriorityQosPolicy transport_priority;
LifespanQosPolicy lifespan;
OwnershipQosPolicy ownership;
};
struct DataWriterQos {
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    DestinationOrderQosPolicy destination_order;
    HistoryQosPolicy history;
    ResourceLimitsQosPolicy resource_limits;
    TransportPriorityQosPolicy transport_priority;
    LifespanQosPolicy lifespan;
    UserDataQosPolicy user_data;
    DDS_OwnershipQosPolicy ownership;
    OwnershipStrengthQosPolicy ownership_strength;
    WriterDataLifecycleQosPolicy writer_data_lifecycle;
};
struct PublisherQos {
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    GroupDataQosPolicy group_data;
    EntityFactoryQosPolicy entity_factory;
};
struct DataReaderQos {
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    DestinationOrderQosPolicy destination_order;
    HistoryQosPolicy history;
    ResourceLimitsQosPolicy resource_limits;
    UserDataQosPolicy user_data;
    DDS_OwnershipQosPolicy ownership;
    TimeBasedFilterQosPolicy time_based_filter;
    ReaderDataLifecycleQosPolicy reader_data_lifecycle;
};

```

```

struct SubscriberQos {
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    GroupDataQosPolicy group_data;
    EntityFactoryQosPolicy entity_factory;
};
//
struct ParticipantBuiltinTopicData {
    BuiltinTopicKey_t key;
    UserDataQosPolicy user_data;
};
struct TopicBuiltinTopicData {
    BuiltinTopicKey_t key;
    string name;
    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    TransportPriorityQosPolicy transport_priority;
    LifespanQosPolicy lifespan;
    DestinationOrderQosPolicy destination_order;
    HistoryQosPolicy history;
    ResourceLimitsQosPolicy resource_limits;
    OwnershipQosPolicy ownership;
    TopicDataQosPolicy topic_data;
};
struct PublicationBuiltinTopicData {
    BuiltinTopicKey_t key;
    BuiltinTopicKey_t participant_key;
    string topic_name;
    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    LifespanQosPolicy lifespan;
    UserDataQosPolicy user_data;
    OwnershipStrengthQosPolicy ownership_strength;
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    TopicDataQosPolicy topic_data;
    GroupDataQosPolicy group_data;
};
struct SubscriptionBuiltinTopicData {
    BuiltinTopicKey_t key;
    BuiltinTopicKey_t participant_key;
    string topic_name;

```

```

    string type_name;
    DurabilityQosPolicy durability;
    DeadlineQosPolicy deadline;
    LatencyBudgetQosPolicy latency_budget;
    LivelinessQosPolicy liveliness;
    ReliabilityQosPolicy reliability;
    DestinationOrderQosPolicy destination_order;
    UserDataQosPolicy user_data;
    TimeBasedFilterQosPolicy time_based_filter;
    PresentationQosPolicy presentation;
    PartitionQosPolicy partition;
    TopicDataQosPolicy topic_data;
    GroupDataQosPolicy group_data;
};
//
interface Entity {
//   ReturnCode_t
//   set_qos(
//       in EntityQos qos);
//
//   ReturnCode_t
//   get_qos(
//       inout EntityQos qos);
//
//   ReturnCode_t
//   set_listener(
//       in Listener l,
//       in StatusMask mask);
//
//   Listener
//   get_listener();
ReturnCode_t
enable();
StatusCondition
get_statuscondition();
StatusMask
get_status_changes();
};
//
interface DomainParticipant : Entity {
    // Factory interfaces
    Publisher
    create_publisher(
        in PublisherQos qos,
        in PublisherListener a_listener,
        in StatusMask mask);
    ReturnCode_t
    delete_publisher(
        in Publisher p);
    Subscriber

```

```

create_subscriber(
    in SubscriberQos qos,
    in SubscriberListener a_listener,
    in StatusMask mask);
ReturnCode_t
delete_subscriber(
    in Subscriber s);
Subscriber
get_builtin_subscriber();
Topic
create_topic(
    in string topic_name,
    in string type_name,
    in TopicQos qos,
    in TopicListener a_listener,
    in StatusMask mask);
ReturnCode_t
delete_topic(
    in Topic a_topic);
Topic
find_topic(
    in string topic_name,
    in Duration_t timeout);
TopicDescription
lookup_topicdescription(
    in string name);
ContentFilteredTopic
create_contentfilteredtopic(
    in string name,
    in Topic related_topic,
    in string filter_expression,
    in StringSeq expression_parameters);
ReturnCode_t
delete_contentfilteredtopic(
    in ContentFilteredTopic a_contentfilteredtopic);
MultiTopic
create_multitopic(
    in string name,
    in string type_name,
    in string subscription_expression,
    in StringSeq expression_parameters);
ReturnCode_t
delete_multitopic(
    in MultiTopic a_multitopic);
ReturnCode_t
delete_contained_entities();
ReturnCode_t
set_qos(
    in DomainParticipantQos qos);
ReturnCode_t

```

```

get_qos(
    inout DomainParticipantQos qos);
ReturnCode_t
set_listener(
    in DomainParticipantListener a_listener,
    in StatusMask mask);
DomainParticipantListener
get_listener();
ReturnCode_t
ignore_participant(
    in InstanceHandle_t handle);
ReturnCode_t
ignore_topic(
    in InstanceHandle_t handle);
ReturnCode_t
ignore_publication(
    in InstanceHandle_t handle);
ReturnCode_t
ignore_subscription(
    in InstanceHandle_t handle);
DomainId_t
get_domain_id();
ReturnCode_t
assert_liveliness();
ReturnCode_t
set_default_publisher_qos(
    in PublisherQos qos);
ReturnCode_t
get_default_publisher_qos(
    inout PublisherQos qos);
ReturnCode_t
set_default_subscriber_qos(
    in SubscriberQos qos);
ReturnCode_t
get_default_subscriber_qos(
    inout SubscriberQos qos);
ReturnCode_t
set_default_topic_qos(
    in TopicQos qos);
ReturnCode_t
get_default_topic_qos(
    inout TopicQos qos);
boolean
contains_entity(
    in InstanceHandle_t a_handle);
ReturnCode_t
get_current_time(
    inout Time_t current_time);
};
interface DomainParticipantFactory {

```

```

//
//  DomainParticipantFactory
//  get_instance();
//
DomainParticipant
create_participant(
    in DomainId_t domainId,
    in DomainParticipantQos qos,
    in DomainParticipantListener a_listener,
    in StatusMask mask);
ReturnCode_t
delete_participant(
    in DomainParticipant a_participant);
DomainParticipant
lookup_participant(
    in DomainId_t domainId);
ReturnCode_t
set_default_participant_qos(
    in DomainParticipantQos qos);
ReturnCode_t
get_default_participant_qos(
    inout DomainParticipantQos qos);

ReturnCode_t
set_qos(
    in DomainParticipantFactoryQos qos);
ReturnCode_t
get_qos(
    inout DomainParticipantFactoryQos qos);
ReturnCode_t
delete_domain
    (in Domain a_domain);
Domain
lookup_domain
    (in DomainId_t domainId);
ReturnCode_t
    create_persistent_snapshot(
        in string partition_expression,
        in string topic_expression,
        in string URI);
ReturnCode_t
    delete_contained_entities();
};
interface TypeSupport {
//  ReturnCode_t
//  register_type(
//      in DomainParticipant domain,
//      in string type_name);
//
//  string

```

```

//  get_type_name();
};
//
interface TopicDescription {
string
    get_type_name();
string
    get_name();
DomainParticipant
    get_participant();
};
interface Topic : Entity, TopicDescription {
ReturnCode_t
set_qos(
    in TopicQos qos);
ReturnCode_t
get_qos(
    inout TopicQos qos);
ReturnCode_t
set_listener(
    in TopicListener a_listener,
    in StatusMask mask);
TopicListener_ptr
get_listener();
// Access the status
ReturnCode_t
get_inconsistent_topic_status(
    inout InconsistentTopicStatus a_status);
};
interface ContentFilteredTopic : TopicDescription {
string
get_filter_expression();
ReturnCode_t
get_expression_parameters(
    inout StringSeq expression_parameters);
ReturnCode_t
set_expression_parameters(
    in StringSeq expression_parameters);
Topic
get_related_topic();
};
interface MultiTopic : TopicDescription {
string
get_subscription_expression();
ReturnCode_t
get_expression_parameters(
    inout StringSeq expression_parameters);
ReturnCode_t
set_expression_parameters(
    in StringSeq expression_parameters);
};

```

```
};
//
interface Publisher : Entity {
DataWriter
create_datawriter(
    in Topic a_topic,
    in DataWriterQos qos,
    in DataWriterListener a_listener,
    in StatusMask mask);
ReturnCode_t
delete_datawriter(
    in DataWriter a_datawriter);
DataWriter
lookup_datawriter(
    in string topic_name);
ReturnCode_t
delete_contained_entities();
ReturnCode_t
set_qos(
    in PublisherQos qos);
ReturnCode_t
get_qos(
    inout PublisherQos qos);
ReturnCode_t
set_listener(
    in PublisherListener a_listener,
    in StatusMask mask);
PublisherListener
get_listener();
ReturnCode_t
suspend_publications();
ReturnCode_t
resume_publications();
ReturnCode_t
begin_coherent_changes();
ReturnCode_t
end_coherent_changes();
ReturnCode_t
wait_for_acknowledgments(
    in Duration_t max_wait);
DomainParticipant
get_participant();
ReturnCode_t
set_default_datawriter_qos(
    in DataWriterQos qos);
ReturnCode_t
get_default_datawriter_qos(
    inout DataWriterQos qos);
ReturnCode_t
copy_from_topic_qos(
```



```

        inout DataWriterQos a_datawriter_qos,
        in TopicQos a_topic_qos);
};
interface DataWriter : Entity {
// InstanceHandle_t
// register_instance(
//     in Data instance_data);
//
// InstanceHandle_t
// register_instance_w_timestamp(
//     in Data instance_data,
//     in Time_t source_timestamp);
//
// ReturnCode_t
// unregister_instance(
//     in Data instance_data,
//     in InstanceHandle_t handle);
//
// ReturnCode_t
// unregister_instance_w_timestamp(
//     in Data instance_data,
//     in InstanceHandle_t handle,
//     in Time_t source_timestamp);
//
// ReturnCode_t
// write(
//     in Data instance_data,
//     in InstanceHandle_t handle);
//
// ReturnCode_t
// write_w_timestamp(
//     in Data instance_data,
//     in InstanceHandle_t handle,
//     in Time_t source_timestamp);
//
// ReturnCode_t
// dispose(
//     in Data instance_data,
//     in InstanceHandle_t instance_handle);
//
// ReturnCode_t
// dispose_w_timestamp(
//     in Data instance_data,
//     in InstanceHandle_t instance_handle,
//     in Time_t source_timestamp);
//
// ReturnCode_t
// get_key_value(
//     inout Data key_holder,
//     in InstanceHandle_t handle);

```

```

//
// InstanceHandle_t lookup_instance(
//     in Data instance_data);
ReturnCode_t
set_qos(
    in DataWriterQos qos);
ReturnCode_t
get_qos(
    inout DataWriterQos qos);
ReturnCode_t
set_listener(
    in DataWriterListener a_listener,
    in StatusMask mask);
DataWriterListener
get_listener();
Topic
get_topic();
Publisher
get_publisher();
ReturnCode_t
wait_for_acknowledgments(
    in Duration_t max_wait);
// Access the status
ReturnCode_t
get_liveliness_lost_status(
    inout LivelinessLostStatus status);
ReturnCode_t
get_offered_deadline_missed_status(
    inout OfferedDeadlineMissedStatus status);
ReturnCode_t
get_offered_incompatible_qos_status(
    inout OfferedIncompatibleQosStatus status);
ReturnCode_t
get_publication_matched_status(
    inout PublicationMatchedStatus status);
ReturnCode_t
    assert_liveliness();
ReturnCode_t
    get_matched_subscriptions(
        inout InstanceHandleSeq subscription_handles);
ReturnCode_t
    get_matched_subscription_data(
        inout SubscriptionBuiltinTopicData subscription_data,
        in InstanceHandle_t subscription_handle);
};
//
interface Subscriber : Entity {
DataReader
create_datareader(
    in TopicDescription a_topic,

```

```

        in DataReaderQos qos,
        in DataReaderListener a_listener,
        in StatusMask mask);
ReturnCode_t
delete_datareader(
    in DataReader a_datareader);
ReturnCode_t
delete_contained_entities();
DataReader
lookup_datareader(
    in string topic_name);
ReturnCode_t
get_datareaders(
    inout DataReaderSeq readers,
    in SampleStateMask sample_states,
    in ViewStateMask view_states,
    in InstanceStateMask instance_states);
ReturnCode_t
notify_datareaders();
ReturnCode_t
set_qos(
    in SubscriberQos qos);
ReturnCode_t
get_qos(
    inout SubscriberQos qos);
ReturnCode_t
set_listener(
    in SubscriberListener a_listener,
    in StatusMask mask);
SubscriberListener
get_listener();
ReturnCode_t
begin_access();
ReturnCode_t
end_access();
DomainParticipant
get_participant();
ReturnCode_t
set_default_datareader_qos(
    in DataReaderQos qos);
ReturnCode_t
get_default_datareader_qos(
    inout DataReaderQos qos);
ReturnCode_t
copy_from_topic_qos(
    inout DataReaderQos a_datareader_qos,
    in TopicQos a_topic_qos);
};
interface DataReader : Entity {
// ReturnCode_t

```

```
// read(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in SampleStateMask sample_states,
//     in ViewStateMask view_states,
//     in InstanceStateMask instance_states);
//
// ReturnCode_t
// take(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in SampleStateMask sample_states,
//     in ViewStateMask view_states,
//     in InstanceStateMask instance_states);
//
// ReturnCode_t
// read_w_condition(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in ReadCondition a_condition);
//
// ReturnCode_t
// take_w_condition(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in ReadCondition a_condition);
//
// ReturnCode_t
// read_next_sample(
//     inout Data data_values,
//     inout SampleInfo sample_info);
//
// ReturnCode_t
// take_next_sample(
//     inout Data data_values,
//     inout SampleInfo sample_info);
//
// ReturnCode_t
// read_instance(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in InstanceHandle_t a_handle,
//     in SampleStateMask sample_states,
//     in ViewStateMask view_states,
//     in InstanceStateMask instance_states);
```

```
//
// ReturnCode_t
// take_instance(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in InstanceHandle_t a_handle,
//     in SampleStateMask sample_states,
//     in ViewStateMask view_states,
//     in InstanceStateMask instance_states);
//
// ReturnCode_t
// read_next_instance(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in InstanceHandle_t a_handle,
//     in SampleStateMask sample_states,
//     in ViewStateMask view_states,
//     in InstanceStateMask instance_states);
//
// ReturnCode_t
// take_next_instance(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in InstanceHandle_t a_handle,
//     in SampleStateMask sample_states,
//     in ViewStateMask view_states,
//     in InstanceStateMask instance_states);
//
// ReturnCode_t
// read_next_instance_w_condition(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in InstanceHandle_t a_handle,
//     in ReadCondition a_condition);
//
// ReturnCode_t
// take_next_instance_w_condition(
//     inout DataSeq data_values,
//     inout SampleInfoSeq info_seq,
//     in long max_samples,
//     in InstanceHandle_t a_handle,
//     in ReadCondition a_condition);
//
// ReturnCode_t
// return_loan(
//     inout DataSeq data_values,
```

```

//      inout SampleInfoSeq info_seq);
//
//  ReturnCode_t
//  get_key_value(
//      inout Data key_holder,
//      in InstanceHandle_t handle);
//
//  InstanceHandle_t
//  lookup_instance(
//      in Data instance);
ReadCondition
create_readcondition(
    in SampleStateMask sample_states,
    in ViewStateMask view_states,
    in InstanceStateMask instance_states);
QueryCondition
create_querycondition(
    in SampleStateMask sample_states,
    in ViewStateMask view_states,
    in InstanceStateMask instance_states,
    in string query_expression,
    in StringSeq query_parameters);
ReturnCode_t
delete_readcondition(
    in ReadCondition a_condition);
ReturnCode_t
delete_contained_entities();
ReturnCode_t
set_qos(
    in DataReaderQos qos);
ReturnCode_t
get_qos(
    inout DataReaderQos qos);
ReturnCode_t
set_listener(
    in DataReaderListener a_listener,
    in StatusMask mask);
DataReaderListener
get_listener();
TopicDescription
get_topicdescription();
Subscriber
get_subscriber();
ReturnCode_t
get_sample_rejected_status(
    inout SampleRejectedStatus status);
ReturnCode_t
get_liveliness_changed_status(
    inout LivelinessChangedStatus status);
ReturnCode_t

```

```

get_requested_deadline_missed_status(
    inout RequestedDeadlineMissedStatus status);
ReturnCode_t
get_requested_incompatible_qos_status(
    inout RequestedIncompatibleQosStatus status);
ReturnCode_t
get_subscription_matched_status(
    inout SubscriptionMatchedStatus status);
ReturnCode_t
get_sample_lost_status(
    inout SampleLostStatus status);
ReturnCode_t
wait_for_historical_data(
    in Duration_t max_wait);
ReturnCode_t
get_matched_publications(
    inout InstanceHandleSeq publication_handles);
ReturnCode_t
get_matched_publication_data(
    inout PublicationBuiltinTopicData publication_data,
    in InstanceHandle_t publication_handle);
};
struct SampleInfo {
SampleStateKind sample_state;
ViewStateKind view_state;
InstanceStateKind instance_state;
Time_t source_timestamp;
InstanceHandle_t instance_handle;
BuiltinTopicKey_t publication_handle;
long disposed_generation_count;
long no_writers_generation_count;
long sample_rank;
long generation_rank;
long absolute_generation_rank;
boolean valid_data;
};
typedef sequence<SampleInfo> SampleInfoSeq;
};
Foo.idl
// Implied IDL for type "Foo"
// Example user defined structure
struct Foo {
long dummy;
};
typedef sequence<Foo> FooSeq;
#include "dds_dcps.idl"
interface FooTypeSupport : DDS::TypeSupport {
DDS::ReturnCode_t
register_type(
    in DDS::DomainParticipant participant,

```

```

        in string type_name);
string
get_type_name();
};
interface FooDataWriter : DDS::DataWriter {
DDS::InstanceHandle_t
register_instance(
    in Foo instance_data);
DDS::InstanceHandle_t
register_instance_w_timestamp(
    in Foo instance_data,
    in DDS::InstanceHandle_t handle,
    in DDS::Time_t source_timestamp);
DDS::ReturnCode_t
unregister_instance(
    in Foo instance_data,
    in DDS::InstanceHandle_t handle);
DDS::ReturnCode_t
unregister_instance_w_timestamp(
    in Foo instance_data,
    in DDS::InstanceHandle_t handle,
    in DDS::Time_t source_timestamp);
DDS::ReturnCode_t
write(
    in Foo instance_data,
    in DDS::InstanceHandle_t handle);
DDS::ReturnCode_t
write_w_timestamp(
    in Foo instance_data,
    in DDS::InstanceHandle_t handle,
    in DDS::Time_t source_timestamp);
DDS::ReturnCode_t
dispose(
    in Foo instance_data,
    in DDS::InstanceHandle_t instance_handle);
DDS::ReturnCode_t
dispose_w_timestamp(
    in Foo instance_data,
    in DDS::InstanceHandle_t instance_handle,
    in DDS::Time_t source_timestamp);
DDS::ReturnCode_t
get_key_value(
    inout Foo key_holder,
    in DDS::InstanceHandle_t handle);
DDS::InstanceHandle_t
lookup_instance(
    in Foo instance_data);
};
interface FooDataReader : DDS::DataReader {
DDS::ReturnCode_t

```



```

read(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::SampleStateMask sample_states,
    in DDS::ViewStateMask view_states,
    in DDS::InstanceStateMask instance_states);
DDS::ReturnCode_t
take(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::SampleStateMask sample_states,
    in DDS::ViewStateMask view_states,
    in DDS::InstanceStateMask instance_states);
DDS::ReturnCode_t
read_w_condition(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::ReadCondition a_condition);
DDS::ReturnCode_t
take_w_condition(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::ReadCondition a_condition);
DDS::ReturnCode_t
read_next_sample(
    inout Foo data_values,
    inout DDS::SampleInfo sample_info);
DDS::ReturnCode_t
take_next_sample(
    inout Foo data_values,
    inout DDS::SampleInfo sample_info);
DDS::ReturnCode_t
read_instance(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::InstanceHandle_t a_handle,
    in DDS::SampleStateMask sample_states,
    in DDS::ViewStateMask view_states,
    in DDS::InstanceStateMask instance_states);
DDS::ReturnCode_t
take_instance(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::InstanceHandle_t a_handle,

```

```

        in DDS::SampleStateMask sample_states,
        in DDS::ViewStateMask view_states,
        in DDS::InstanceStateMask instance_states);
DDS::ReturnCode_t
read_next_instance(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::InstanceHandle_t a_handle,
    in DDS::SampleStateMask sample_states,
    in DDS::ViewStateMask view_states,
    in DDS::InstanceStateMask instance_states);
DDS::ReturnCode_t
take_next_instance(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::InstanceHandle_t a_handle,
    in DDS::SampleStateMask sample_states,
    in DDS::ViewStateMask view_states,
    in DDS::InstanceStateMask instance_states);
DDS::ReturnCode_t
read_next_instance_w_condition(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::InstanceHandle_t a_handle,
    in DDS::ReadCondition a_condition);
DDS::ReturnCode_t
take_next_instance_w_condition(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq,
    in long max_samples,
    in DDS::InstanceHandle_t a_handle,
    in DDS::ReadCondition a_condition);
DDS::ReturnCode_t
return_loan(
    inout FooSeq data_values,
    inout DDS::SampleInfoSeq info_seq);
DDS::ReturnCode_t
get_key_value(
    inout Foo key_holder,
    in DDS::InstanceHandle_t handle);
DDS::InstanceHandle_t
lookup_instance(
    in Foo instance);
};

```

D *SampleStates, ViewStates and InstanceStates*

Data is made available to the application by the following operations on `DDS_DataReader` objects: `DDS_DataReader_read` and `DDS_DataReader_take` operations. The general semantics of the `DDS_DataReader_read` operations is that the application only gets access to the matching data; the data remain available in the Data Distribution Services and can be read again. The semantics of the `DDS_DataReader_take` operations is that the data is not available in the Data Distribution Service; that data will no longer be accessible to the `DDS_DataReader`. Consequently, it is possible for a `DDS_DataReader` to access the same sample multiple times but only if all previous accesses were `DDS_DataReader_read` operations.

Each of these operations returns an ordered collection of Data values and associated `DDS_SampleInfo` objects. Each data value represents an atom of data information (i.e., a value for one instance). This collection may contain samples related to the same or different instances (identified by the key). Multiple samples can refer to the same instance if the settings of the `DDS_HistoryQosPolicy` allow for it.

SampleInfo Class

`DDS_SampleInfo` is the information that accompanies each sample that is ‘read’ or ‘taken’. It contains, among others, the following information:

- The `sample_state` (`DDS_READ_SAMPLE_STATE` or `DDS_NOT_READ_SAMPLE_STATE`);
- The `view_state`, (`DDS_NEW_VIEW_STATE` or `DDS_NOT_NEW_VIEW_STATE`);
- The `instance_state` (`DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` or `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`).

sample_state

For each sample, the Data Distribution Service internally maintains a `sample_state` specific to each `DDS_DataReader`. The `sample_state` can either be `DDS_READ_SAMPLE_STATE` or `DDS_NOT_READ_SAMPLE_STATE`.

- `DDS_READ_SAMPLE_STATE` indicates that the `DDS_DataReader` has already accessed that sample by means of `DDS_DataReader_read`. Had the sample been accessed by `DDS_DataReader_take` it would no longer be available to the `DDS_DataReader`;
- `DDS_NOT_READ_SAMPLE_STATE` indicates that the `DDS_DataReader` has not accessed that sample before.

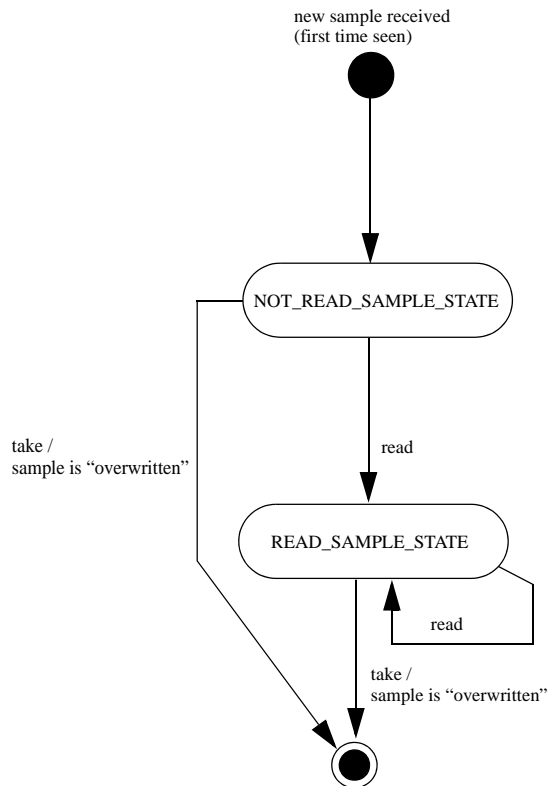


Figure 20: State Chart of the sample_state for a Single Sample

State per Sample

The `sample_state` available in the `DDS_SampleInfo` reflect the `sample_state` of each sample. The `sample_state` can be different for all samples in the returned collection that refer to the same instance.

instance_state

For each instance the Data Distribution Service internally maintains an `instance_state`. The `instance_state` can be:

- `DDS_ALIVE_INSTANCE_STATE` indicates that:
 - samples have been received for the instance
 - there are live `DDS_DataWriter` objects writing the instance
 - the instance has not been explicitly disposed of (or else samples have been received after it was disposed of)
- `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` indicates the instance was disposed of by a `DDS_DataWriter` either explicitly by means of the `DDS_DataWriter_dispose` operation or implicitly in case the `autodispose_unregistered_instances` field of the `WriterDataLifecycleQosPolicy` equals `TRUE` when the instance gets unregistered (see Section 3.1.3.23, *DDS_WriterDataLifecycleQosPolicy*), and no new samples for that instance have been written afterwards.
- `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE` indicates the instance has been declared as not-alive by the `DDS_DataReader` because it detected that there are no live `DDS_DataWriter` objects writing that instance.

DDS_OwnershipQosPolicy

The precise events that cause the `instance_state` to change depends on the setting of the `DDS_OwnershipQosPolicy`:

- If `DDS_OwnershipQosPolicy` is set to `DDS_EXCLUSIVE_OWNERSHIP_QOS`, then the `instance_state` becomes `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` only if the `DDS_DataWriter` that “owns” the instance explicitly disposes of it. The `instance_state` becomes `DDS_ALIVE_INSTANCE_STATE` again only if the `DDS_DataWriter` that owns the instance writes it;
- If `DDS_OwnershipQosPolicy` is set to `DDS_SHARED_OWNERSHIP_QOS`, then the `instance_state` becomes `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` if any `DDS_DataWriter` explicitly disposes of the instance. The `instance_state` becomes `DDS_ALIVE_INSTANCE_STATE` as soon as any `DDS_DataWriter` writes the instance again.

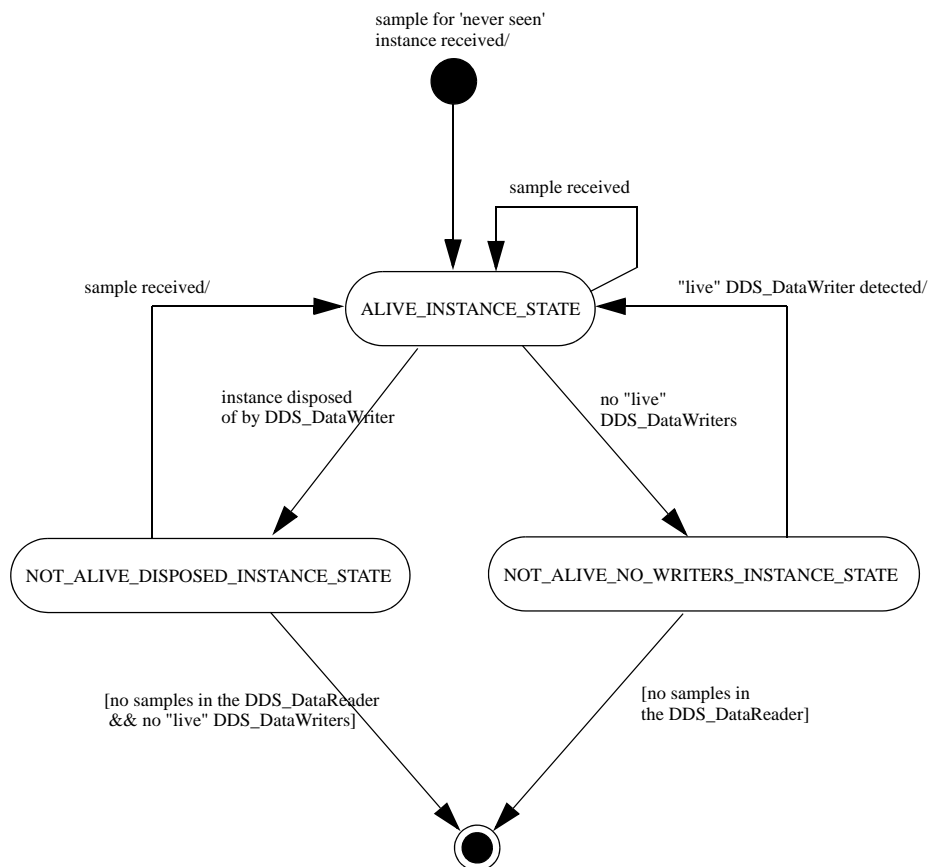


Figure 21: State Chart of the instance_state for a Single Instance

Snapshot

The `instance_state` available in the `DDS_SampleInfo` is a snapshot of the `instance_state` of the instance at the time the collection was obtained (i.e. at the time `DDS_DataReader_read` or `DDS_DataReader_take` was called). The `instance_state` is therefore the same for all samples in the returned collection that refer to the same instance.

view_state

For each instance (identified by the key), the Data Distribution Service internally maintains a `view_state` relative to each `DDS_DataReader`. The `view_state` can either be `DDS_NEW_VIEW_STATE` or `DDS_NOT_NEW_VIEW_STATE`.

- `DDS_NEW_VIEW_STATE` indicates that either this is the first time that the `DDS_DataReader` has ever accessed samples of that instance, or else that the `DDS_DataReader` has accessed previous samples of the instance, but the instance has since been reborn (i.e. becomes not-alive and then alive again);
- `DDS_NOT_NEW_VIEW_STATE` indicates that the `DDS_DataReader` has already accessed samples of the same instance and that the instance has not been reborn since.

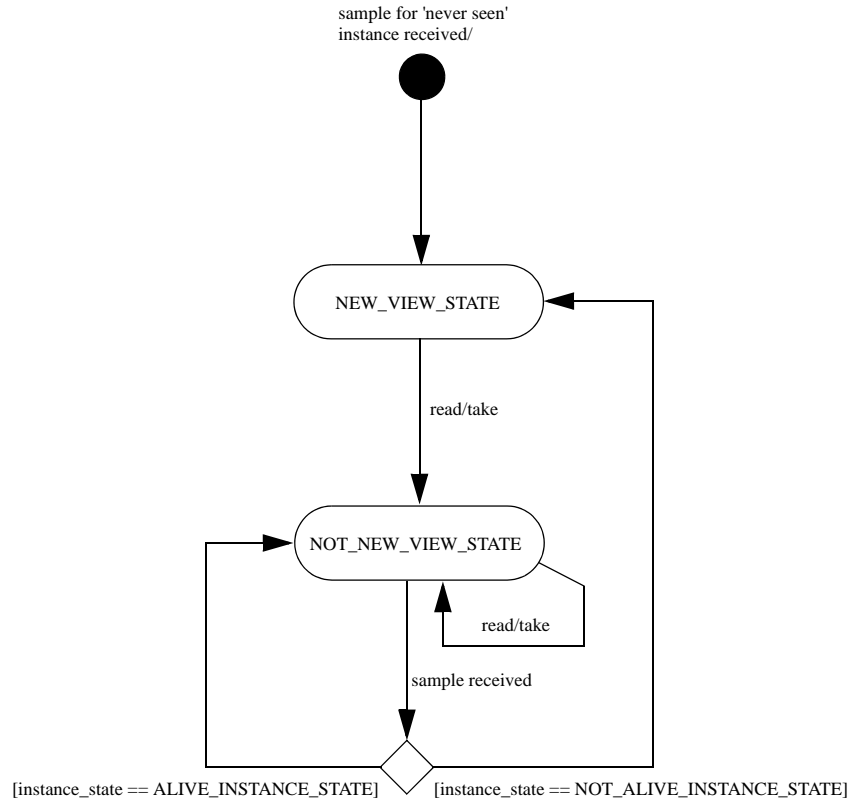


Figure 22: State Chart of the `view_state` for a Single Instance

Snapshot

The `view_state` available in the `DDS_SampleInfo` is a snapshot of `view_state` of the instance relative to the `DDS_DataReader` used to access the samples at the time the collection was obtained (i.e. at the time `DDS_DataReader_read` or `DDS_DataReader_take` was called). The `view_state` is therefore the same for all samples in the returned collection that refer to the same instance.

State Masks

State Definitions

All states are available as a constant. These convenience constants can be used to create a bit-mask (e.g. to be used as operation parameters) by performing an AND or OR operation. They can also be used for testing whether a state is set.

The sample state definitions indicates whether or not the matching data sample has already been read:

- `DDS_READ_SAMPLE_STATE`: sample has already been read;
- `DDS_NOT_READ_SAMPLE_STATE`: sample has not been read.

The view state definitions indicates whether the `DDS_DataReader` has already seen samples for the most-current generation of the related instance:

- `DDS_NEW_VIEW_STATE`: all samples of this instance are new;
- `DDS_NOT_NEW_VIEW_STATE`: some or all samples of this instance are not new.

The instance state definitions indicates whether the instance is currently in existence or, if it has been disposed of, the reason why it was disposed of:

- `DDS_ALIVE_INSTANCE_STATE`: this instance is currently in existence;
- `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`: this instance was disposed of by a `DDS_DataWriter`;
- `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`: the instance has been disposed of by the `DDS_DataReader` because none of the `DDS_DataWriter` objects currently “alive” (according to the `DDS_LivelinessQosPolicy`) are writing the instance.

Pre-defined Bit Mask Definitions

For convenience, some pre-defined bit-masks are available as a constant definition. These bit-mask constants can be used where a state bit-mask is required. They can also be used for testing whether certain bits are set.

The sample state bit-mask definition selects both sample states:

- `DDS_ANY_SAMPLE_STATE`: either the sample has already been read or not read;

The view state bit-mask definition selects both view states:

- `DDS_ANY_VIEW_STATE`: either the sample has already been seen or not seen;

The instance state bit-mask definitions selects a combination of instance states:

- `DDS_NOT_ALIVE_INSTANCE_STATE`: this instance was disposed of by a `DDS_DataWriter` or the `DDS_DataReader`;
- `DDS_ANY_INSTANCE_STATE`: this instance is either in existence or not in existence.

Operations Concerning States

The application accesses data by means of the operations `DDS_DataReader_read` or `DDS_DataReader_take` on the `DDS_DataReader`. These operations return an ordered collection of `DDS_DataSamples` consisting of a `DDS_SampleInfo` part and a `Data` part. The way the Data Distribution Service builds this collection (i.e., the data-samples that are parts of the list as well as their order) depends on `QosPolicy` settings set on the `DDS_DataReader` and the `DDS_Subscriber`, as well as the source timestamp of the samples and the parameters passed to the `DDS_DataReader_read/DDS_DataReader_take` operations, namely:

- the desired sample states (i.e., `DDS_READ_SAMPLE_STATE`, `DDS_NOT_READ_SAMPLE_STATE`, or `DDS_ANY_SAMPLE_STATE`);
- the desired view states (i.e., `DDS_NEW_VIEW_STATE`, `DDS_NOT_NEW_VIEW_STATE`, or `DDS_ANY_VIEW_STATE`);
- the desired instance states (`DDS_ALIVE_INSTANCE_STATE`, `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`, `DDS_NOT_ALIVE_INSTANCE_STATE`, or `DDS_ANY_INSTANCE_STATE`).

The `DDS_DataReader_read` and `DDS_DataReader_take` operations are non-blocking and just deliver what is currently available that matches the specified states.

On output, the collection of `Data` values and the collection of `DDS_SampleInfo` structures are of the same length and are in a one-to-one correspondence. Each `DDS_SampleInfo` provides information, such as the `source_timestamp`, the `sample_state`, `view_state`, and `instance_state`, etc., about the matching sample.

Some elements in the returned collection may not have valid data. If the `instance_state` in the `DDS_SampleInfo` is `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` or `DDS_NOT_ALIVE_NO_WRITERS_INSTANCE_STATE`, then the last sample for that instance in the collection, that is, the one whose `DDS_SampleInfo` has `sample_rank==0` does not contain valid data. Samples that contain no data do not count towards the limits imposed by the `DDS_ResourceLimitsQosPolicy`.

read

The act of reading a sample sets its `sample_state` to `DDS_READ_SAMPLE_STATE`. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to `DDS_NOT_NEW_VIEW_STATE`. It will not affect the `instance_state` of the instance.

take

The act of taking a sample removes it from the `DDS_DataReader` so it cannot be ‘read’ or ‘taken’ again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to `DDS_NOT_NEW_VIEW_STATE`. It will not affect the `instance_state` of the instance.

read_w_condition

In case the `DDS_ReadCondition` is a ‘plain’ `DDS_ReadCondition` and not the specialized `DDS_QueryCondition`, the operation is equivalent to calling `DDS_DataReader_read` and passing as `sample_states`, `view_states` and `instance_states` the value of the corresponding attributes in the `DDS_ReadCondition`. Using this operation the application can avoid repeating the same parameters specified when creating the `DDS_ReadCondition`.

take_w_condition

The act of taking a sample removes it from the `DDS_DataReader` so it cannot be ‘read’ or ‘taken’ again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to `DDS_NOT_NEW_VIEW_STATE`. It will not affect the `instance_state` of the instance.

In case the `DDS_ReadCondition` is a ‘plain’ `DDS_ReadCondition` and not the specialized `DDS_QueryCondition`, the operation is equivalent to calling `DDS_DataReader_take` and passing as `sample_states`, `view_states` and `instance_states` the value of the corresponding attributes in the `DDS_ReadCondition`. Using this operation the application can avoid repeating the same parameters specified when creating the `DDS_ReadCondition`.

read_next_sample

The `DDS_DataReader_read_next_sample` operation is semantically equivalent to the `DDS_DataReader_read` operation where the input Data sequence has `max_len=1`, the `sample_states=DDS_NOT_READ_SAMPLE_STATE`, the `view_states=DDS_ANY_VIEW_STATE`, and the `instance_states=DDS_ANY_INSTANCE_STATE`.

take_next_sample

The `DDS_DataReader_take_next_sample` operation is semantically equivalent to the `DDS_DataReader_take` operation where the input sequence has `max_len=1`, the `sample_states=DDS_NOT_READ_SAMPLE_STATE`, the `view_states=DDS_ANY_VIEW_STATE`, and the `instance_states=DDS_ANY_INSTANCE_STATE`.

read_instance

The act of reading a sample sets its `sample_state` to `DDS_READ_SAMPLE_STATE`. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to `DDS_NOT_NEW_VIEW_STATE`. It will not affect the `instance_state` of the instance.

take_instance

The act of taking a sample removes it from the `DDS_DataReader` so it cannot be ‘read’ or ‘taken’ again. If the sample belongs to the most recent generation of the instance, it will also set the `view_state` of the instance to `DDS_NOT_NEW_VIEW_STATE`. It will not affect the `instance_state` of the instance.

Appendix

E Class Inheritance

This appendix gives an overview of the inheritance relations of the DCPS classes.

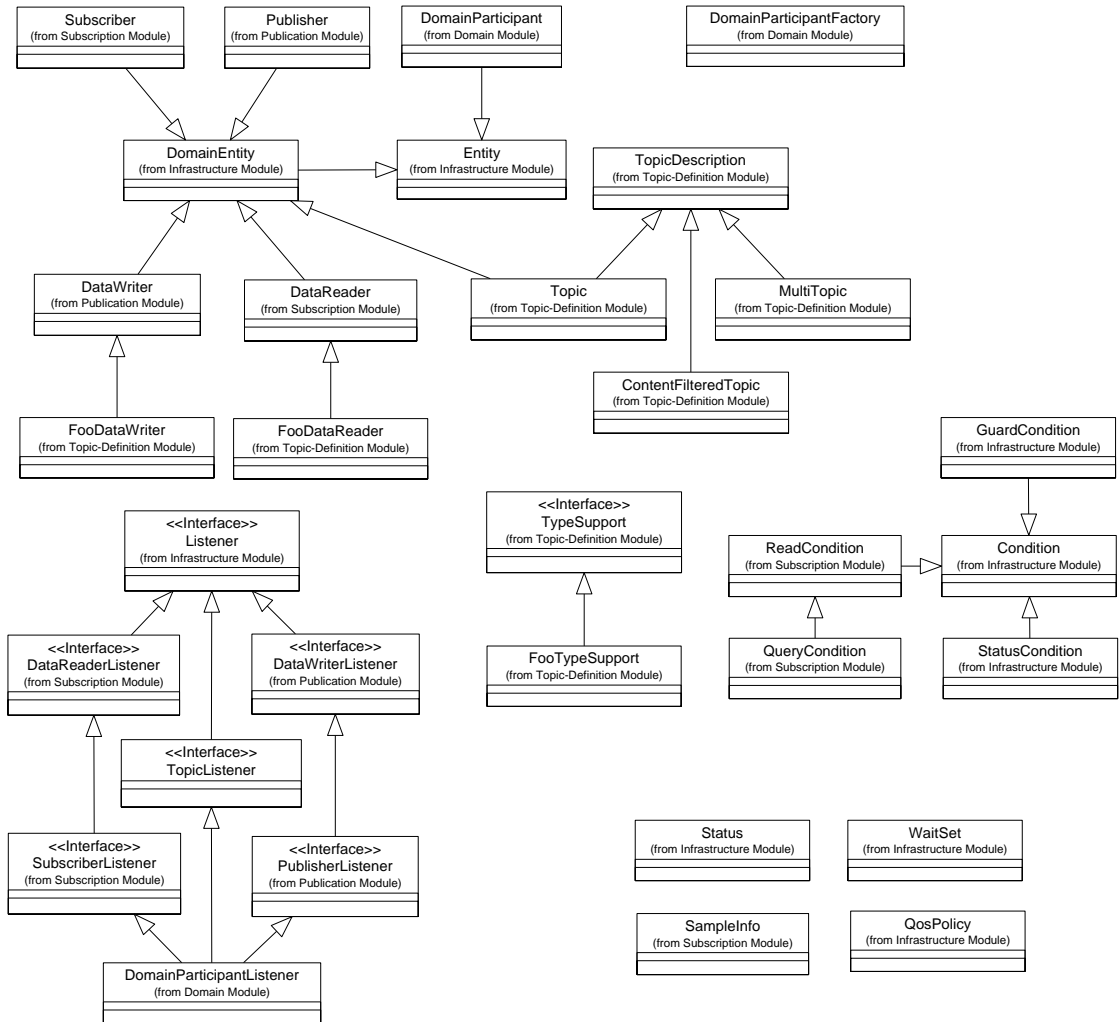


Figure 23 DCPS Inheritance

F

Listeners, Conditions and Waitsets

`Listeners` and `DDS_Conditions` (`DDS_Conditions` in conjunction with `DDS_WaitSets`) are two mechanisms that allow the application to be made aware of changes in the communication status. `Listeners` provide an event-based mechanism for the Data Distribution Service to asynchronously alert the application of the occurrence of relevant status changes. `DDS_Conditions` in conjunction with `DDS_WaitSets` provide a state-based mechanism for the Data Distribution Service to synchronously communicate the relevant status changes to the application.

Both mechanisms are based on the communication statuses associated with an `DDS_Entity` object. Not all statuses are applicable to all `DDS_Entity` objects. Which status is applicable to which `DDS_Entity` object is listed in the next table:

Table 25 Communication Status

DDS_Entity	Status Name	Description
DDS_Topic	DDS_INCONSISTENT_TOPIC_STATUS	Another <code>DDS_Topic</code> exists with the same name but with different characteristics.
DDS_Subscriber	DDS_DATA_ON_READERS_STATUS	New information is available.

Table 25 Communication Status (Continued)

DDS_Entity	Status Name	Description
DDS_DataReader	DDS_SAMPLE_REJECTED_STATUS	A (received) sample has been rejected.
	DDS_LIVELINESS_CHANGED_STATUS	The liveliness of one or more DDS_DataWriter objects, that were writing instances read through the DDS_DataReader objects has changed. Some DDS_DataWriter object have become “alive” or “not alive”.
	DDS_REQUESTED_DEADLINE_MISSED_STATUS	The deadline that the DDS_DataReader was expecting through its DDS_DeadlineQosPolicy was not respected for a specific instance.
	DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS	A QosPolicy setting was incompatible with what is offered.
	DDS_DATA_AVAILABLE_STATUS	New information is available.
	DDS_SAMPLE_LOST_STATUS	A sample has been lost (never received).
	DDS_SUBSCRIPTION_MATCHED_STATUS	The DDS_DataReader has found a DDS_DataWriter that matches the ZWDDS_Topic and has compatible QoS.
DDS_DataWriter	DDS_LIVELINESS_LOST_STATUS	The liveliness that the DDS_DataWriter has committed through its DDS_LivelinessQosPolicy was not respected; thus DDS_DataReader objects will consider the DDS_DataWriter as no longer “alive”.
	DDS_OFFERED_DEADLINE_MISSED_STATUS	The deadline that the DDS_DataWriter has committed through its DDS_DeadlineQosPolicy was not respected for a specific instance.
	DDS_OFFERED_INCOMPATIBLE_QOS_STATUS	A QosPolicy setting was incompatible with what was requested.
	DDS_PUBLICATION_MATCHED_STATUS	The DDS_DataWriter has found DDS_DataReader that matches the DDS_Topic and has compatible QoS.

The statuses may be classified in:

- *read communication statuses*: i.e., those that are related to arrival of data, namely DDS_DATA_ON_READERS and DDS_DATA_AVAILABLE
- *plain communication statuses*: i.e., all the others.

For each plain communication status, there is a corresponding status struct. The information from this struct can be retrieved with the operations `get_<status_name>_status`. For example, to get the `DDS_INCONSISTENT_TOPIC` status (which information is stored in the `DDS_InconsistentTopicStatus` struct), the application must call the operation `DDS_Topic_get_inconsistent_topic_status`. A plain communication status can only be read from the `DDS_Entity` on which it is applicable. For the read communication statuses there is no struct available to the application.

Communication Status Event

Conceptually associated with each `DDS_Entity` communication status is a logical `StatusChangedFlag`. This flag indicates whether that particular communication status has changed since the last time the status was ‘read’ by the application (there is no actual read-operation to read the `StatusChangedFlag`). The `StatusChangedFlag` is only conceptually needed to explain the behaviour of a `Listener`, therefore, it is not important whether this flag actually exists. A `Listener` will only be activated when the `StatusChangedFlag` changes from `FALSE` to `TRUE` (provided the `Listener` is attached and enabled for this particular status). The conditions which cause the `StatusChangedFlag` to change is slightly different for the plain communication status and the read communication status.

For the plain communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` whenever the plain communication status changes and it is reset to `FALSE` each time the application accesses the plain communication status via the proper `get_<status_name>_status` operation on the `DDS_Entity`.

The communication status is also reset to `FALSE` whenever the associated `Listener` operation is called as the `Listener` implicitly accesses the status which is passed as a parameter to the operation. The fact that the status is reset prior to calling the listener means that if the application calls the `get_<status_name>_status` from inside the listener it will see the status already reset.

An exception to this rule is when the associated `Listener` is the *nil* listener, in other word, a listener with value `DDS_OBJECT_NIL`. Such a listener is treated as a NOOP¹ for all statuses activated in its bitmask and the act of calling this ‘nil’ listener does not reset the corresponding communication statuses.

1. Short for **No-Operation**, an instruction that performs nothing at all.

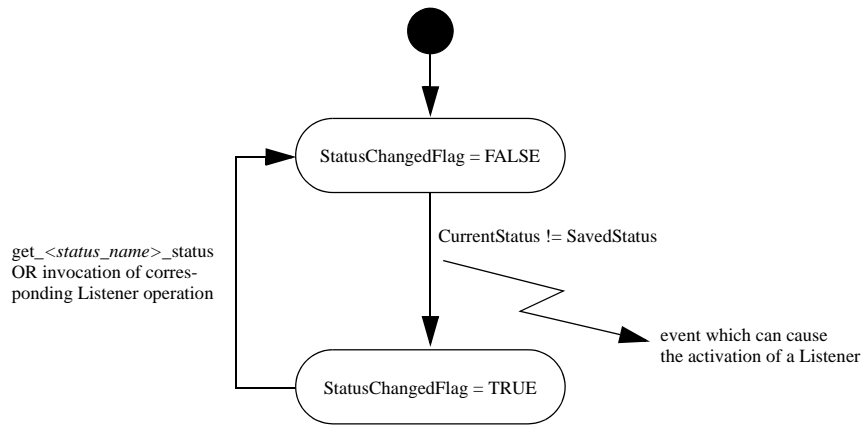


Figure 24: Plain Communication Status State Chart

For example, the value of the `StatusChangedFlag` associated with the `DDS_RequestedDeadlineMissedStatus` will become `TRUE` each time a new deadline passes (which increases the `total_count` field within `DDS_RequestedDeadlineMissedStatus`). The value changes to `FALSE` when the application accesses the status via the corresponding `DDS_DataReader_get_requested_deadline_missed_status` operation on the proper `DDS_Entity`, or when the `on_requested_deadline_missed` operation on the `Listener` attached to this `DDS_Entity` or one its containing entities is invoked.

For the read communication status, the `StatusChangedFlag` flag is initially set to `FALSE`. It becomes `TRUE` when data arrives, or when the `InstanceState` of a contained instance changes. This can be caused by either:

- The arrival of the notification that an instance has been disposed by:
 - the `DDS_DataWriter` that owns it if its `OwnershipQosPolicyKind = DDS_EXCLUSIVE_OWNERSHIP_QOS`
 - or by any `DDS_DataWriter` if its `OwnershipQosPolicyKind = DDS_SHARED_OWNERSHIP_QOS`.
- The loss of liveness of the `DDS_DataWriter` of an instance for which there is no other `DDS_DataWriter`.
- The arrival of the notification that an instance has been unregistered by the only `DDS_DataWriter` that is known to be writing the instance.

The read communication statuses are reset to `FALSE` again in the following circumstances:

- The status flag of the `DDS_DATA_AVAILABLE_STATUS` becomes `FALSE` when either the corresponding listener operation (`on_data_available`) is called, or the read or take operation (or any of its variants) is called on the associated `DDS_DataReader`.

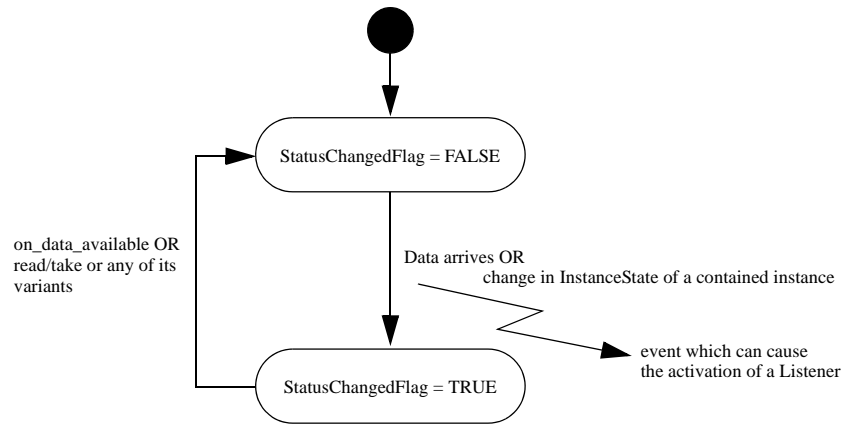


Figure 25: Read Communication Status `DDS_DataReader` Statecraft

- The status flag of the `DDS_DATA_ON_READERS_STATUS` becomes `FALSE` when any of the following events occurs:
 - The corresponding listener operation (`on_data_on_readers`) is called on the corresponding `DDS_Subscriber`.
 - The `on_data_available` listener operation is called on any `DDS_DataReader` belonging to the `DDS_Subscriber`.
 - The read or take operation (or any of its variants) is called on any `DDS_DataReader` belonging to the `DDS_Subscriber`.

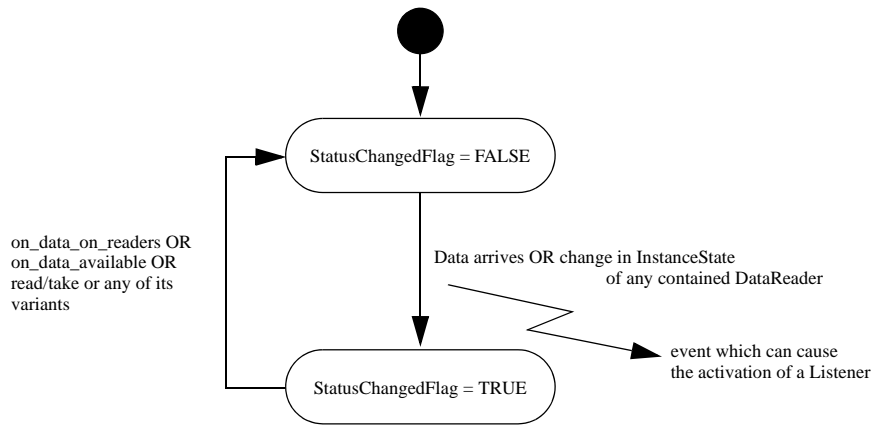


Figure 26: DDS_Subscriber Statecraft for a Read Communication Status

Listeners

The `Listeners` provide for an event-based mechanism to asynchronously inform the application of a status change event. `Listeners` are applicable for both the read communication statuses and the plain communication statuses. When one of these status change events occur, the associated `Listener` is activated, provided some pre-conditions are satisfied. When the `Listener` is activated, it will call the corresponding `on_<status_name>` operation of that `Listener`. Each `on_<status_name>` operation available in the `Listener` of an `DDS_Entity` is also available in the `Listener` of the factory of the `DDS_Entity`.

For both the read communication statuses and the plain communication statuses a `Listener` is only activated when a `Listener` is attached to this particular `DDS_Entity` and enabled for this particular status. Statuses are enabled according to the `DDS_StatusKindMask` parameter that was passed at creation time of the `DDS_Entity`, or that was passed to the `DDS_<DDS_Entity>_set_listener` operation .

When an event occurs for a particular `DDS_Entity` and for a particular status, but the applicable `Listener` is not activated for this status, the status is propagated up to the factory of this `DDS_Entity`. For this factory, the same propagation rules apply. When even the `DDS_DomainParticipantListener` is not attached or enabled for this status, the application will not be notified about this event. This means that a status change on a contained `DDS_Entity` only invokes the `Listener` of its factory if the `Listener` of the contained `DDS_Entity` itself does not handle the trigger event generated by the status change.

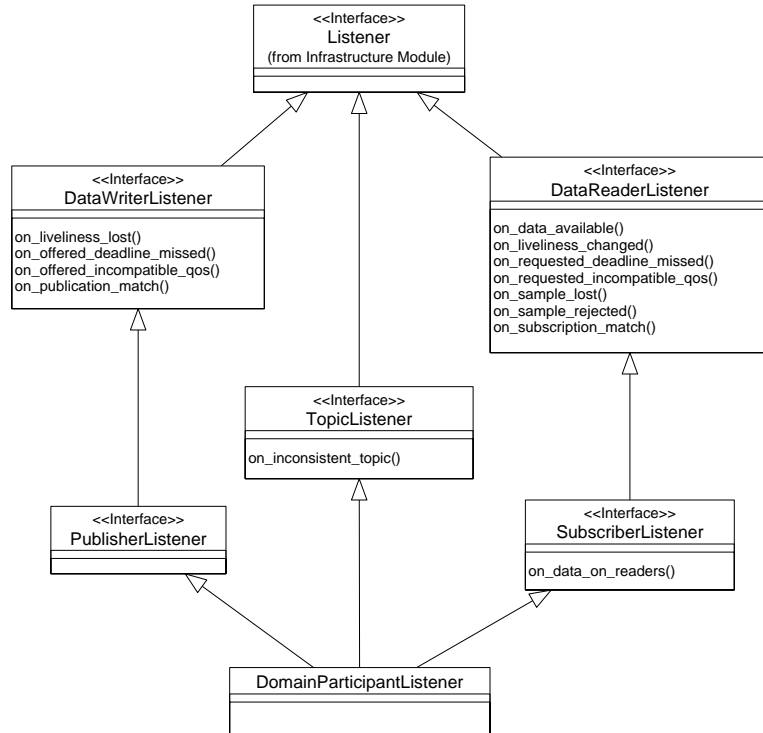


Figure 27: DCPS Listeners

The event propagation is also applicable to the read communication statuses. However, since the event here is the arrival of data, both the DDS_DATA_ON_READERS and DDS_DATA_AVAILABLE status are TRUE. The Data Distribution Service will first attempt to handle the DDS_DATA_ON_READERS status and try to activate the DDS_SubscriberListener. When this Listener is not activated for this status, the event will propagate to the DDS_DomainParticipantListener. Only when the DDS_DATA_ON_READERS status can not be handled, the Data Distribution Service will attempt to handle the DDS_DATA_AVAILABLE status and try to activate the DDS_DataReaderListener. In case this Listener is not activated for this status, the event will follow the propagation rules as described above.

Conditions and Waitsets

The DDS_Conditions in conjunction with DDS_WaitSets provide for a wait-based mechanism to synchronously inform the application of status changes. A DDS_Condition can be either a DDS_ReadCondition, DDS_QueryCondition, DDS_StatusCondition or DDS_GuardCondition. To create a DDS_Condition one of the following operations can be used:

- `DDS_ReadCondition` created by
`DDS_DataReader_create_readcondition`
- `DDS_QueryCondition` created by
`DDS_DataReader_create_querycondition`
- `DDS_StatusCondition` retrieved by
`DDS_<Entity>_get_statuscondition` on an `DDS_<Entity>`
- `DDS_GuardCondition` created by the C operation
`DDS_GuardCondition__alloc`

Note that the `DDS_QueryCondition` is a specialized `DDS_ReadCondition`. The `DDS_GuardCondition` is a different kind of `DDS_Condition` since it is not controlled by a status but directly by the application (when a `DDS_GuardCondition` is initially created, the `trigger_value` is `FALSE`). The `DDS_StatusCondition` is present by default with each `DDS_Entity`, therefore, it does not have to be created.

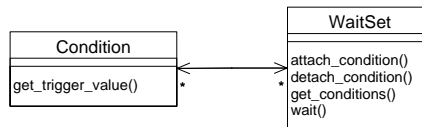


Figure 28: DCPS DDS_WaitSets

A `DDS_WaitSet` may have one or several `DDS_Conditions` attached to it. An application thread may block execution (blocking may be limited by a timeout) by waiting on a `DDS_WaitSet` until the `trigger_value` of one or more of the `DDS_Conditions` become `TRUE`. When a `DDS_Condition`, whose `trigger_value` evaluates to `TRUE`, is attached to a `DDS_WaitSet` that is currently being waited on (using the `DDS_WaitSet_wait` operation), the `DDS_WaitSet` will unblock immediately.

This (wait-based) mechanism is generally used as follows:

- The application creates a `DDS_WaitSet`
- The application indicates which relevant information it wants to be notified of, by creating or retrieving `DDS_Condition` objects (`DDS_StatusCondition`, `DDS_ReadCondition`, `DDS_QueryCondition` or `DDS_GuardCondition`) and attach them to a `DDS_WaitSet`
- It then waits on that `DDS_WaitSet` (using `DDS_WaitSet_wait`) until the `trigger_value` of one or several `DDS_Condition` objects (in the `DDS_WaitSet`) become `TRUE`

- When the thread is unblocked, the application uses the result of the `DDS_WaitSet_wait` (i.e., the list of `DDS_Condition` objects with `trigger_value==TRUE`) to actually get the information:
 - if the condition is a `DDS_StatusCondition` and the status changes refer to a plain communication status, by calling `get_status_changes` and then `get_<communication_status>` on the relevant `DDS_Entity`
 - if the condition is a `DDS_StatusCondition` and the status changes refer to the read communication status:
 - `DDS_DATA_ON_READERS`, by calling `get_status_changes` and then `DDS_Subscriber_get_datareaders` on the relevant `DDS_Subscriber` and then `DDS_DataReader_read/DDS_DataReader_take` on the returned `DDS_DataReader` objects
 - `DDS_DATA_AVAILABLE`, by calling `get_status_changes` and then `DDS_DataReader_read/DDS_DataReader_take` on the relevant `DDS_DataReader`.
 - if it is a `DDS_ReadCondition` or a `DDS_QueryCondition`, by calling directly `DDS_DataReader_read_w_condition` / `DDS_DataReader_take_w_condition` on the `DDS_DataReader` with the `DDS_Condition` as a parameter.

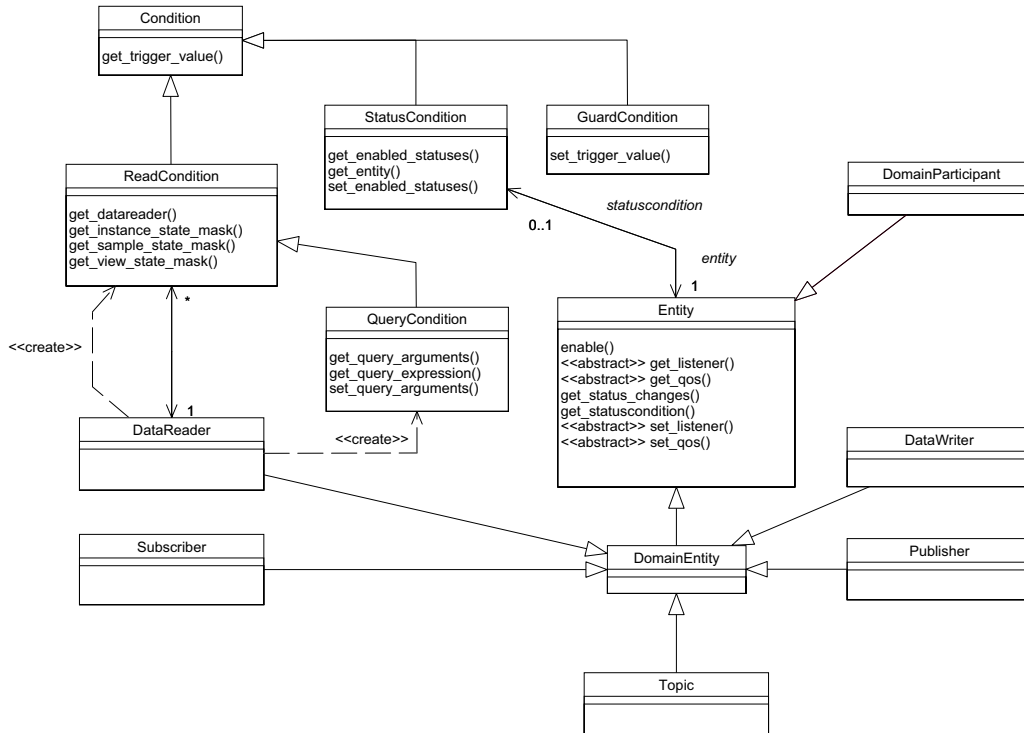


Figure 29 DCPS DDS_Conditions

No extra information is passed from the Data Distribution Service to the application when a `DDS_WaitSet_wait` returns only the list of triggered `DDS_Condition` objects. Therefore, it is the application responsibility to investigate which `DDS_Condition` objects have triggered the `DDS_WaitSet`.

Blocking Behaviour

The result of a `DDS_WaitSet_wait` operation depends on the state of the `DDS_WaitSet`, which in turn depends on whether at least one attached `DDS_Condition` has a `trigger_value` of `TRUE`. If the `DDS_WaitSet_wait` operation is called on `DDS_WaitSet` with state `BLOCKED` it will block the calling thread. If `DDS_WaitSet_wait` is called on a `DDS_WaitSet` with state `UNBLOCKED` it will return immediately. In addition, when the `DDS_WaitSet` transitions from state `BLOCKED` to state `UNBLOCKED` it wakes up the thread (if any) that had called `DDS_WaitSet_wait` on it. Note that there can only be one thread waiting on a single `DDS_WaitSet`.

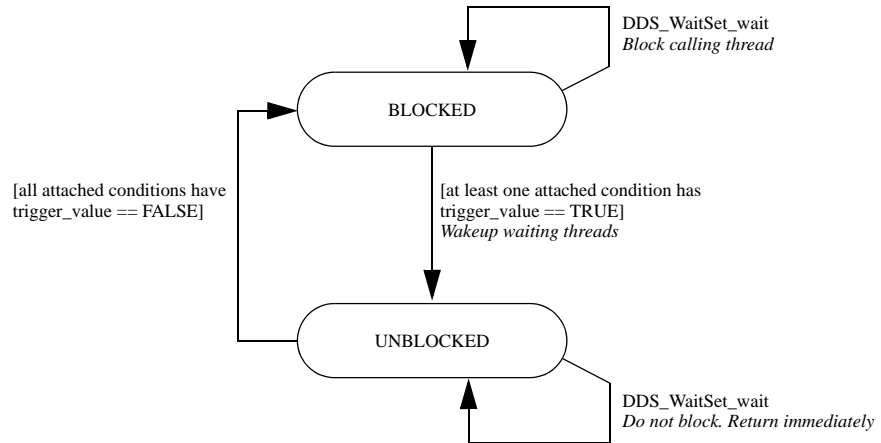


Figure 30: Blocking Behaviour of a Waitset State Chart

DDS_StatusCondition Trigger State

The `trigger_value` of a `DDS_StatusCondition` is the boolean OR of the `StatusChangedFlag` of all the communication statuses to which it is sensitive. That is, `trigger_value == FALSE` only if all the values of the `StatusChangedFlags` are `FALSE`.

The sensitivity of the `DDS_StatusCondition` to a particular communication status is controlled by the bit-mask of `enabled_statuses` set on the `DDS_Condition` by means of the `DDS_StatusCondition_set_enabled_statuses` operation.

DDS_ReadCondition and DDS_QueryCondition Trigger State

Similar to the `DDS_StatusCondition`, a `DDS_ReadCondition` also has a `trigger_value` that determines whether the attached `DDS_WaitSet` is `BLOCKED` or `UNBLOCKED`. However, unlike the `DDS_StatusCondition`, the `trigger_value` of the `DDS_ReadCondition` is tied to the presence of at least one sample managed by the Data Distribution Service with `SampleState`, `ViewState`, and `InstanceState` matching those of the `DDS_ReadCondition`. Additionally, for the `DDS_QueryCondition`, the data associated with the sample, must be such that the `query_expression` evaluates to `TRUE`.

The fact that the `trigger_value` of a `DDS_ReadCondition` is dependent on the presence of samples on the associated `DDS_DataReader` implies that a single `DDS_DataReader_take` operation can potentially change the `trigger_value` of several `DDS_ReadCondition` or `DDS_QueryCondition` objects. For example, if all samples are taken, any `DDS_ReadCondition` or `DDS_QueryCondition`

objects associated with the `DDS_DataReader` that had their `trigger_value==TRUE` before will see the `trigger_value` change to `FALSE`. Note that this does not guarantee that `DDS_WaitSet` objects, that had those `DDS_Condition` objects separately attached to, will not be woken up. Once we have `trigger_value==TRUE` on a `DDS_Condition` it may wake up the `DDS_WaitSet` it was attached to, the condition transitions to `trigger_value==FALSE` does not 'un-wake up' the `DDS_WaitSet` as 'un-wakening' is not possible. The consequence is that an application blocked on a `DDS_WaitSet` may return from the wait with a list of `DDS_Condition` objects some of which are no longer “active”. This is unavoidable if multiple threads are concurrently waiting on separate `DDS_WaitSet` objects and taking data associated with the same `DDS_DataReader DDS_Entity`. In other words, a `DDS_WaitSet_wait` may return with a list of `DDS_Condition` objects which all have a `trigger_value==FALSE`. This only means that at some point one or more of the `DDS_Condition` objects have had a `trigger_value==TRUE` but no longer do.

DDS_GuardCondition Trigger State

The `trigger_value` of a `DDS_GuardCondition` is completely controlled by the application via the operation `DDS_GuardCondition_set_trigger_value`. This `DDS_Condition` can be used to implement an application defined wake-up of the blocked thread.

G *DDS_Topic Definitions*

The Data Distribution Service distributes its data in structured data types, called topics. The first step when using the Data Distribution Service consists of defining these topics. Since the Data Distribution Service supports using several programming languages, OMG IDL is used for this purpose. This appendix describes how to define the topics.

DDS_Topic Definition Example

All data distributed using the Data Distribution Service has to be defined as a topic. A topic is a structured data type, like a C-struct with several members. Whenever the application needs to read or write data, it will be reading or writing topics. The definition of each topic it will be using has to be written in (a subset of) OMG IDL. For example:

```
module SPACE {
    struct Foo {
        long      userID; // owner of message
        long long index;  // message index per owner
        string     content; // message body
    };
    #pragma keylist Foo
};
```

This is the definition of a topic called `Foo`, used for sending and receiving messages (as an example). Even though the topic is defined using IDL, the Data Distribution Service will be using an equivalent C-struct which is accessed by the application using the type specific operations. Generation of the typed classes is achieved by invoking the Data Distribution Service IDL pre-processor: `idlpp -l c -S <idl_filename>.idl`, a tool which translates the IDL topic definition into an equivalent C definition. The `-l c` option indicates that C-code has to be generated, the `-S` option indicates that this C code should be StandAlone C code, i.e. it must not have any dependency on external ORB libraries. In this example, the pre-processor will generate the classes `SPACE_FooTypeSupport`, `SPACE_FooDataWriter` and `SPACE_FooDataReader` which contain the type specific operations.

The prefix `SPACE_` is generated from the IDL-module-name. The types of the fields are prescribed by the IDL-to-C mapping. After the Data Distribution Service IDL-pre-processor is run, the application will use the generated code.

Complex Topics

The `Foo` topic is relatively simple, but the Data Distribution Service is capable of distributing more complex topics as well. In fact, any definition following the OpenSplice IDL subset is allowed. For a reference of this subset, see the BNF-notation in Appendix , *Data Distribution Service IDL Subset in BNF Notation*. It is important to know that the pre-processor accepts all IDL constructs but only the subset is being processed.

Apart from the trivial data types, the Data Distribution Service is capable of handling fixed-length arrays, bounded and unbounded sequences, union types and enumerations. Types can be nested, e.g. a struct can contain a struct field or an array of structs, or a sequence of strings or an array of sequences containing structs. For more information regarding the IDL to C mapping.

IDL Pre-processor

This section contains the specification of the subset of OMG IDL that can be used to define the topics.

IDL to Host Language Mapping

The Data Distribution Service IDL pre-processor translates the IDL-definition of the topics into language specific code. This translation is executed according to the OMG IDL mappings. Since the Data Distribution Service uses data-structures only, not all IDL-features are implemented by the pre-processor. Usually, the IDL definition consists of a module defining several structs and typedefs.

Data Distribution Service IDL Keywords

The identifiers listed in this appendix are reserved for use as keywords in IDL and may not be used otherwise, unless escaped with a leading underscore.

abstract	exception	inout	provides	truncatable
any	emits	interface	public	typedef
attribute	enum	local	publishes	typeid
boolean	eventtype	long	raises	typeprefix
case	factory	module	readonly	unsigned
char	FALSE	multiple	setraises	union
component	finder	native	sequence	uses
const	fixed	Object	short	ValueBase
consumes	float	octet	string	valuetype
context	getraises	oneway	struct	void

custom	home	out	supports	wchar
default	import	primarykey	switch	wstring
double	in	private	TRUE	

Keywords must be written exactly as shown in the above list. Identifiers that collide with keywords are illegal. For example, `boolean` is a valid keyword; `Boolean` and `BOOLEAN` are illegal identifiers.

Data Distribution Service IDL Pragma Keylist

To define a topic, the content must either be a struct or a union. The pre-processor will only generate the type specific classes when topic definition is accompanied by a `<pragmakeylist>`. When the `<pragmakeylist>` has no `<field_id>`, the topic is available but no key is set. To define the keylist the definition, written in BNF-notation, is as follows:

```

<pragmakeylist> ::= "#pragma keylist" <type_id>
<field_id>*
<type_id> ::= <struct_type_identifier>
            | <union_type_identifier>
<field_id> ::= <member_declarator>
            | <element_spec_declarator>

```

In case of a struct, `<type_id>` is a `<struct_type_identifier>`. In case of a union, `<type_id>` is a `<union_type_identifier>`. The `<struct_type_identifier>` is the identifier used in the struct declaration. The `<union_type_identifier>` is the identifier used in the union declaration. The `<field_id>` is the identifier of a field in the struct or union identified by `<type_id>`. In case of a struct, `<field_id>` is a `<member_declarator>` which is one of the declarators used in the struct member. In case of a union, `<field_id>` is a `<element_spec_declarator>` which is one of the declarators used in the element specification in a case of the union.

For example, for the `Foo` example in Appendix , *DDS_Topic Definition Example* the next pragma must be used to have the pre-processor generate the typed classes (`SPACE_FooTypeSupport`, `SPACE_FooDataWriter` and `SPACE_FooDataReader`).

```
#pragma keylist Foo userID index
```

Note that in this example the `userID` and the `index` are used as a key.

Data Distribution Service IDL Subset in BNF Notation

Only a subset is used by the pre-processor. A description of the Data Distribution Service IDL subset, written in BNF-notation, is as follows:

```

<definition> ::= <type_dcl> ";"
              | <const_dcl> ";"

```

```

    | <module> ";"
<module> ::= "module" <identifier> "{" <definition>+ "}"
<scoped_name> ::= <identifier>
    | "::" <identifier>
    | <scoped_name> "::" <identifier>
<const_dcl> ::= "const" <const_type>
    <identifier> "=" <const_exp>
<const_type> ::= <integer_type>
    | <char_type>
    | <boolean_type>
    | <floating_pt_type>
    | <string_type>
    | <scoped_name>
    | <octet_type>
<const_exp> ::= <or_expr>
<or_expr> ::= <xor_expr>
    | <or_expr> "|" <xor_expr>
<xor_expr> ::= <and_expr>
    | <xor_expr> "^" <and_expr>
<and_expr> ::= <shift_expr>
    | <and_expr> "&" <shift_expr>
<shift_expr> ::= <add_expr>
    | <shift_expr> ">" <add_expr>
    | <shift_expr> "<" <add_expr>
<add_expr> ::= <mult_expr>
    | <add_expr> "+" <mult_expr>
    | <add_expr> "-" <mult_expr>
<mult_expr> ::= <unary_expr>
    | <mult_expr> "*" <unary_expr>
    | <mult_expr> "/" <unary_expr>
    | <mult_expr> "%" <unary_expr>
<unary_expr> ::= <unary_operator> <primary_expr>
    | <primary_expr>
<unary_operator> ::= "-"
    | "+"
    | "~"
<primary_expr> ::= <scoped_name>
    | <literal>
    | "(" <const_exp> ")"
<literal> ::= <integer_literal>
    | <string_literal>
    | <character_literal>
    | <floating_pt_literal>
    | <boolean_literal>
<boolean_literal> ::= "TRUE"
    | "FALSE"
<positive_int_const> ::= <const_exp>
<type_dcl> ::= "typedef" <type_declarator>
    | <struct_type>
    | <union_type>

```

```

    | <enum_type>
<type_declarator>::= <type_spec> <declarators>
<type_spec>::= <simple_type_spec>
    | <constr_type_spec>
<simple_type_spec>::= <base_type_spec>
    | <template_type_spec>
    | <scoped_name>
<base_type_spec>::= <floating_pt_type>
    | <integer_type>
    | <char_type>
    | <boolean_type>
    | <octet_type>
<template_type_spec>::= <sequence_type>
    | <string_type>
<constr_type_spec>::= <struct_type>
    | <union_type>
    | <enum_type>
<declarators>::= <declarator> { "," <declarator> }*
<declarator>::= <simple_declarator>
    | <complex_declarator>
<simple_declarator>::= <identifier>
<complex_declarator>::= <array_declarator>
<floating_pt_type>::= "float"
    | "double"
<integer_type>::= <signed_int>
    | <unsigned_int>
<signed_int>::= <signed_short_int>
    | <signed_long_int>
    | <signed_longlong_int>
<signed_short_int>::= "short"
<signed_long_int>::= "long"
<signed_longlong_int>::= "long" "long"
<unsigned_int>::= <unsigned_short_int>
    | <unsigned_long_int>
    | <unsigned_longlong_int>
<unsigned_short_int>::= "unsigned" "short"
<unsigned_long_int>::= "unsigned" "long"
<unsigned_longlong_int>::= "unsigned" "long" "long"
<char_type>::= "char"
<boolean_type>::= "boolean"
<octet_type>::= "octet"
<struct_type>::= "struct" <identifier> "{" <member_list> "}"
<member_list>::= <member>+
<member>::= <type_spec> <declarators> ";"
<union_type>::= "union" <identifier> "switch"
    "(" <switch_type_spec> ")"
    "{" <switch_body> "}"
<switch_type_spec>::= <integer_type>
    | <char_type>
    | <boolean_type>

```

```

        | <enum_type>
        | <scoped_name>
<switch_body>::= <case>+
<case>::= <case_label>+ <element_spec> ";"
<case_label>::= "case" <const_exp> ":"
        | "default" ":"
<element_spec>::= <type_spec> <declarator>
<enum_type>::= "enum" <identifier>
        "{" <enumerator> { "," <enumerator> }* "}"
<enumerator>::= <identifier>
<sequence_type>::= "sequence" "<" <simple_type_spec> ","
        <positive_int_const> ">"
        | "sequence" "<" <simple_type_spec> ">"
<string_type>::= "string" "<" <positive_int_const> ">"
        | "string"
<array_declarator>::= <identifier> <fixed_array_size>+
<fixed_array_size>::= "[" <positive_int_const> "]"

```


H DCPS Queries and Filters

A subset of SQL syntax is used in several parts of OpenSplice:

- the `filter_expression` in the `DDS_ContentFilteredTopic`
- the `topic_expression` in the `DDS_MultiTopic`
- the `query_expression` in the `DDS_QueryReadCondition`.

Those expressions may use a subset of SQL, extended with the possibility to use program variables in the SQL expression. The allowed SQL expressions are defined with the BNF-grammar below. The following notational conventions are made:

- the NonTerminals are typeset in *italics*
- the 'Terminals' are quoted and typeset in a fixed width font
- the TOKENS are typeset in small caps
- the notation (*element* // ',') represents a non-empty comma-separated list of *elements*.

SQL Grammar in BNF

```

Expression ::= FilterExpression
            | TopicExpression
            | QueryExpression
FilterExpression ::= Condition
TopicExpression ::= SelectFrom {Where } ';'
QueryExpression ::= {Condition} {'ORDER BY' (FIELDNAME // ',')}
SelectFrom ::= 'SELECT' Aggregation 'FROM' Selection
Aggregation ::= '*'
            | (SubjectFieldSpec // ',')
SubjectFieldSpec ::= FIELDNAME
                | FIELDNAME 'AS' FIELDNAME
                | FIELDNAME FIELDNAME
Selection ::= TOPICNAME
            | TOPICNAME NaturalJoin JoinItem
JoinItem ::= TOPICNAME
            | TOPICNAME NaturalJoin JoinItem
            | '(' TOPICNAME NaturalJoin JoinItem ')'
NaturalJoin ::= 'INNER NATURAL JOIN'
            | 'NATURAL JOIN'
            | 'NATURAL INNER JOIN'
Where ::= 'WHERE' Condition
Condition ::= Predicate
            | Condition 'AND' Condition

```

```

        | Condition 'OR' Condition
        | 'NOT' Condition
        | '(' Condition ')'
Predicate ::= ComparisonPredicate
        | BetweenPredicate
ComparisonPredicate ::= FIELDNAME RelOp Parameter
        | Parameter RelOp FIELDNAME

BetweenPredicate ::= FIELDNAME 'BETWEEN' Range
        | FIELDNAME 'NOT BETWEEN' Range
RelOp ::= '=' | '>' | '>=' | '<' | '<=' | '<>' | like
Range ::= Parameter 'AND' Parameter
Parameter ::= INTEGERVALUE
        | FLOATVALUE
        | STRING
        | ENUMERATEDVALUE
        | PARAMETER

```

Note: INNER NATURAL JOIN, NATURAL JOIN, and NATURAL INNER JOIN are all aliases, in the sense that they have the same semantics. The aliases are all supported because they all are part of the SQL standard.

SQL Token Expression

The syntax and meaning of the tokens used in the SQL grammar is described as follows:

FIELDNAME - A fieldname is a reference to a field in the data-structure. The dot '.' is used to navigate through nested structures. The number of dots that may be used in a fieldname is unlimited. The field-name can refer to fields at any depth in the data structure. The names of the field are those specified in the IDL definition of the corresponding structure, which may or may not match the field names that appear on the C mapping of the structure

TOPICNAME - A topic name is an identifier for a topic, and is defined as any series of characters 'a', ..., 'z', 'A', ..., 'Z', '0', ..., '9', '-', '_' but may not start with a digit

INTEGERVALUE - Any series of digits, optionally preceded by a plus or minus sign, representing a decimal integer value within the range of the system. A hexadecimal number is preceded by 0x and must be a valid hexadecimal expression

FLOATVALUE - Any series of digits, optionally preceded by a plus or minus sign and optionally including a floating point ('.'). A power-of-ten expression may be post-fixed, which has the syntax en, where n is a number, optionally preceded by a plus or minus sign

STRING - Any series of characters encapsulated in single quotes, except a new-line character or a right quote. A string starts with a left or right quote, but ends with a right quote

ENUMERATEDVALUE - An enumerated value is a reference to a value declared within an enumeration. The name of the value must correspond to the names specified in the IDL definition of the enumeration, and must be encapsulated in single quotes. An enum value starts with a left or right quote, but ends with a right quote.

PARAMETER - A parameter is of the form $\%n$, where n represents a natural number (zero included) smaller than 100. It refers to the $n + 1^{\text{th}}$ argument in the given context.

Note: when RelOp is 'like', Unix filename wildcards must be used for strings instead of the normal SQL wildcards. This means any one character is '?', any zero or more characters is '*'

SQL Examples

Assuming Topic "Location" has as an associated type a structure with fields "flight_name, x, y, z", and Topic "FlightPlan" has as fields "flight_id, source, destination". The following are examples of using these expressions.

Example of a *topic_expression*:

"SELECT flight_name, x, y, z AS height FROM 'Location' NATURAL JOIN 'FlightPlan' WHERE height < 1000 AND x < 23".

Example of a *query_expression* or a *filter_expression*:

"height < 1000 AND x < 23".

A close-up, low-angle photograph of a computer keyboard, focusing on the central and right-hand keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

BIBLIOGRAPHY

Bibliography

- [1] *OMG Data Distribution Service Revised Final Adopted Specification ptc/04-03-07*, Object Management Group
- [2] *OMG C Language Mapping Specification formal/99-07-35*, Object Management Group (OMG)
- [3] *OMG The Common Object Request Broker: Architecture and Specification, Version 3.0, formal/02-06-01*, Object Management Group

A close-up, low-angle photograph of a computer keyboard, focusing on the central and right-hand keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

GLOSSARY

Glossary

Acronyms

<i>Acronym</i>	<i>Meaning</i>
CORBA	Common Object Request Broker Architecture
DCPS	Data Centric Publish/Subscribe
DDS	Data Distribution Service
DLRL	Data Local Reconstruction Layer
IDL	Interface Definition Language
OMG	Object Management Group
ORB	Object Request Broker
QoS	Quality of Service
SPLICE	Subscription Paradigm for the Logical Interconnection of Concurrent Engines

A close-up, low-angle photograph of a computer keyboard, focusing on the central and right-hand keys. The keys are white with dark lettering. A white grid pattern is overlaid on the entire image, creating a sense of depth and perspective. The lighting is soft, highlighting the texture of the keys and the grid lines.

INDEX

Index

A

About the C Reference Guide. xxi Affected Entities. 481

B

Basic Usage 481 Blocking Behaviour 550
Bibliography. 565 Blocking Behaviour of a Waitset State Chart . 551

C

C Reference Guide Document Structure 3
Changeable 61
Class DDS_Condition 126
Class DDS_ContentFilteredTopic 224
Class DDS_DataReader (abstract) 371
Class DDS_DataSample 448
Class DDS_DataWriter (abstract) 272
Class DDS_DomainEntity (abstract) 55
Class DDS_DomainParticipant 136
Class DDS_DomainParticipantFactory 185
Class DDS_Entity (abstract). 48
Class DDS_GuardCondition 127
Class DDS_MultiTopic 230
Class DDS_Publisher 245
Class DDS_QueryCondition 471
Class DDS_ReadCondition 467
Class DDS_StatusCondition 130
Class DDS_Subscriber 347
Class DDS_Topic 215
Class DDS_TopicDescription (abstract) 212
Class DDS_TypeSupport (abstract). 238
Class DDS_WaitSet 119
Class Model of the DCPS Topic-Definition
Module 211
Class SPACE_FooDataReader. 409
Class SPACE_FooDataWriter 300
Class SPACE_FooTypeSupport. 239
Communication Status 541
Communication Status Event. 543
Complex Topics 554
Conditions and waitsets 547
Contacts xxiii

D

Data Distribution Service Defined Type 10
Data Distribution Service IDL Keywords 554
Data Distribution Service IDL Pragma Keylist 555
Data Distribution Service IDL Subset in BNF
Notation. 555
Data Type “Foo” Typed Classes Pre-processor
Generation. 42
DCPS DDS_Conditions 126, 550
DCPS DDS_Status Values 105
DCPS DDS_WaitSets. 119, 548
DCPS Domain Module’s Class Model. 40, 136
DCPS Infrastructure Module’s Class Model 38, 48
DCPS Inheritance. 539
DCPS Listeners 102, 547
DCPS Module Composition. 37
DCPS Publication Module’s Class Model. 43
DCPS Subscription Module’s Class Model. 44
DCPS Topic-Definition Module’s Class Model 41
DDS__alloc 19
DDS__alloc 17
DDS__alloc 19
DDS__allocbuf. 18
DDS_Condition_get_trigger_value 127
DDS_ContentFilteredTopic_get_expression_para

- meters. 225
- DDS_ContentFilteredTopic_get_filter_expression 226
- DDS_ContentFilteredTopic_get_name (inherited) 227
- DDS_ContentFilteredTopic_get_participant (inherited). 227
- DDS_ContentFilteredTopic_get_related_topic 227
- DDS_ContentFilteredTopic_get_type_name (inherited). 228
- DDS_ContentFilteredTopic_set_expression_parameters. 228
- DDS_DataReader 545
- DDS_DataReader_create_querycondition . . . 377
- DDS_DataReader_create_readcondition . . . 378
- DDS_DataReader_delete_contained_entities . 380
- DDS_DataReader_delete_readcondition . . . 381
- DDS_DataReader_enable (inherited) 382
- DDS_DataReader_get_key_value (abstract). . 382
- DDS_DataReader_get_listener 383
- DDS_DataReader_get_liveliness_changed_status 383
- DDS_DataReader_get_matched_publication_data 384
- DDS_DataReader_get_matched_publications 386
- DDS_DataReader_get_qos 388
- DDS_DataReader_get_requested_deadline_missed_status. 389
- DDS_DataReader_get_requested_incompatible_qos_status. 390
- DDS_DataReader_get_sample_lost_status. . . 391
- DDS_DataReader_get_sample_rejected_status 393
- DDS_DataReader_get_status_changes (inherited) 394
- DDS_DataReader_get_statuscondition (inherited) 394
- DDS_DataReader_get_subscriber. 394
- DDS_DataReader_get_subscription_match_status 395
- DDS_DataReader_get_topicdescription 396
- DDS_DataReader_lookup_instance (abstract) 291, 397
- DDS_DATAREADER_QOS_DEFAULT . . . 485
- DDS_DataReader_read (abstract). 397
- DDS_DataReader_read_instance (abstract) . . 398
- DDS_DataReader_read_next_instance (abstract). 398
- DDS_DataReader_read_next_instance_w_condition (abstract). 399
- DDS_DataReader_read_next_sample (abstract). . 399
- DDS_DataReader_read_w_condition (abstract). . 399
- DDS_DataReader_return_loan (abstract). . . . 400
- DDS_DataReader_set_listener 400
- DDS_DataReader_set_qos 403
- DDS_DataReader_take (abstract). 405
- DDS_DataReader_take_instance (abstract) . . 405
- DDS_DataReader_take_next_instance (abstract). 405
- DDS_DataReader_take_next_instance_w_condition (abstract). 406
- DDS_DataReader_take_next_sample (abstract). . 406
- DDS_DataReader_take_w_condition (abstract). . 407
- DDS_DataReader_wait_for_historical_data. . 407
- DDS_DataReaderListener. 34
- DDS_DataReaderListener interface 458
- DDS_DataReaderListener_alloc. 460
- DDS_DataReaderListener_on_data_available (abstract) 461
- DDS_DataReaderListener_on_liveliness_changed (abstract) 462
- DDS_DataReaderListener_on_requested_deadline_missed (abstract) 463
- DDS_DataReaderListener_on_requested_incompatible_qos (abstract). 464
- DDS_DataReaderListener_on_sample_lost (abstract) 465
- DDS_DataReaderListener_on_sample_rejected (abstract) 465
- DDS_DataReaderListener_on_subscription_matched (abstract). 466
- DDS_DataReaderQos 483
- DDS_DataWriter_assert_liveliness 276
- DDS_DataWriter_dispose (abstract). 278
- DDS_DataWriter_dispose_w_timestamp (abstract) 278, 300
- DDS_DataWriter_enable (inherited) 278

- DDS_DataWriter_get_key_value (abstract) . . . 279
- DDS_DataWriter_get_listener 279
- DDS_DataWriter_get_liveliness_lost_status . . 279
- DDS_DataWriter_get_matched_subscription_data
281
- DDS_DataWriter_get_matched_subscriptions. 282
- DDS_DataWriter_get_offered_deadline_missed_s
tatus 284
- DDS_DataWriter_get_offered_incompatible_qos_
status 285
- DDS_DataWriter_get_publication_match_status .
287
- DDS_DataWriter_get_publisher 288
- DDS_DataWriter_get_qos 289
- DDS_DataWriter_get_status_changes (inherited) .
290
- DDS_DataWriter_get_statuscondition (inherited) .
290
- DDS_DataWriter_get_topic 290
- DDS_DATAWRITER_QOS_DEFAULT . . . 488
- DDS_DataWriter_register_instance (abstract). 291
- DDS_DataWriter_register_instance_w_timestamp
(abstract) 292
- DDS_DataWriter_set_listener 292
- DDS_DataWriter_set_qos 294
- DDS_DataWriter_unregister_instance (abstract) . .
296
- DDS_DataWriter_unregister_instance_w_timesta
mp (abstract) 296
- DDS_DataWriter_write (abstract) 299
- DDS_DataWriter_write_w_timestamp (abstract) .
299
- DDS_DataWriterListener 31
- DDS_DataWriterListener interface 340
- DDS_DataWriterListener__alloc 341
- DDS_DataWriterListener_on_liveliness_lost
(abstract) 342
- DDS_DataWriterListener_on_offered_deadline_
missed (abstract) 343
- DDS_DataWriterListener_on_offered_incompatib
le_qos (abstract) 344
- DDS_DataWriterListener_on_publication_match
(abstract) 345
- DDS_DataWriterQos 486
- dds_dcps.idl 501
- DDS_DeadlineQosPolicy 63, 65, 78
- DDS_DestinationOrderQosPolicy 66
- DDS_DomainParticipant_assert_liveliness . . 141
- DDS_DomainParticipant_contains_entity . . . 142
- DDS_DomainParticipant_create_contentfilteredto
pic 143
- DDS_DomainParticipant_create_multitopic . . 144
- DDS_DomainParticipant_create_publisher . . 146
- DDS_DomainParticipant_create_subscriber . . 149
- DDS_DomainParticipant_create_topic 152
- DDS_DomainParticipant_delete_contained_entiti
es 154
- DDS_DomainParticipant_delete_contentfilteredto
pic 156
- DDS_DomainParticipant_delete_multitopic . . 158
- DDS_DomainParticipant_delete_publisher . . 159
- DDS_DomainParticipant_delete_subscriber . . 160
- DDS_DomainParticipant_delete_topic 161
- DDS_DomainParticipant_enable (inherited) . . 163
- DDS_DomainParticipant_find_topic 163
- DDS_DomainParticipant_get_builtin_subscriber .
165
- DDS_DomainParticipant_get_current_time . . 165
- DDS_DomainParticipant_get_default_publisher_
qos 167
- DDS_DomainParticipant_get_default_subscriber_
qos 168
- DDS_DomainParticipant_get_default_topic_qos .
170
- DDS_DomainParticipant_get_discovered_particip
ant_data 171
- DDS_DomainParticipant_get_discovered_particip
ants 171
- DDS_DomainParticipant_get_discovered_topic_d
ata 172
- DDS_DomainParticipant_get_discovered_topics .
171
- DDS_DomainParticipant_get_domain_id . . . 172
- DDS_DomainParticipant_get_listener 173
- DDS_DomainParticipant_get_qos 173
- DDS_DomainParticipant_get_status_changes
(inherited) 174
- DDS_DomainParticipant_get_statuscondition
(inherited) 175
- DDS_DomainParticipant_ignore_participant . 175

DDS_DomainParticipant_ignore_publication	175	DDS_DomainParticipantListener_on_requested_incompatible_qos (inherited, abstract) . . .	209
DDS_DomainParticipant_ignore_subscription	175	DDS_DomainParticipantListener_on_sample_lost (inherited, abstract)	210
DDS_DomainParticipant_ignore_topic	176	DDS_DomainParticipantListener_on_sample_rejected (inherited, abstract)	210
DDS_DomainParticipant_lookup_topicdescription	176	DDS_DomainParticipantListener_on_subscription_match (inherited, abstract)	210
DDS_DomainParticipant_set_default_publisher_qos	177	DDS_DomainParticipantQos	488
DDS_DomainParticipant_set_default_subscriber_qos	178	DDS_DurabilityQosPolicy	67
DDS_DomainParticipant_set_default_topic_qos	179	DDS_DurabilityServiceQosPolicy	71
DDS_DomainParticipant_set_listener	181	DDS_Entity_enable	49
DDS_DomainParticipant_set_qos	184	DDS_Entity_get_instance_handle	51
DDS_DomainParticipantFactory_create_participant	187	DDS_Entity_get_listener (abstract)	52
DDS_DomainParticipantFactory_delete_participant	190	DDS_Entity_get_qos (abstract)	52
DDS_DomainParticipantFactory_get_default_participant_qos	191	DDS_Entity_get_status_changes	53
DDS_DomainParticipantFactory_get_instance	193	DDS_Entity_get_statuscondition	54
DDS_DomainParticipantFactory_lookup_participant	194	DDS_Entity_set_listener (abstract)	54
DDS_DomainParticipantFactory_set_default_participant_qos	196	DDS_Entity_set_qos (abstract)	55
DDS_DomainParticipantListener	27	DDS_EntityFactoryQosPolicy	73
DDS_DomainParticipantListener interface.	203	DDS_free	21
DDS_DomainParticipantListener_alloc	206	DDS_GroupDataQosPolicy	74
DDS_DomainParticipantListener_on_data_available (inherited, abstract)	207	DDS_GuardCondition_alloc	128
DDS_DomainParticipantListener_on_data_on_readers (inherited, abstract)	207	DDS_GuardCondition_get_trigger_value (inherited)	129
DDS_DomainParticipantListener_on_inconsistent_topic (inherited, abstract)	207	DDS_GuardCondition_set_trigger_value	129
DDS_DomainParticipantListener_on_liveliness_changed (inherited, abstract)	208	DDS_HistoryQosPolicy	75
DDS_DomainParticipantListener_on_liveliness_lost (inherited, abstract)	208	DDS_InconsistentTopicStatus	106
DDS_DomainParticipantListener_on_offered_deadline_missed (inherited, abstract)	208	DDS_LatencyBudgetQosPolicy	77
DDS_DomainParticipantListener_on_offered_incompatible_qos (inherited, abstract)	208	DDS_LifespanQosPolicy	79
DDS_DomainParticipantListener_on_publication_match (inherited, abstract)	209	DDS_Listener interface	101
DDS_DomainParticipantListener_on_requested_deadline_missed (inherited, abstract)	209	DDS_LivelinessChangedStatus	107
		DDS_LivelinessLostStatus	109
		DDS_LivelinessQosPolicy	80, 81
		DDS_MultiTopic_get_expression_parameters	231
		DDS_MultiTopic_get_name (inherited)	232
		DDS_MultiTopic_get_participant (inherited)	232
		DDS_MultiTopic_get_subscription_expression	233
		DDS_MultiTopic_get_type_name (inherited)	233
		DDS_MultiTopic_set_expression_parameters	234
		DDS_OfferedDeadlineMissedStatus	110
		DDS_OfferedIncompatibleQosStatus	111
		DDS_OwnershipQosPolicy	82, 531
		DDS_OwnershipStrengthQosPolicy	84

- DDS_PARTICIPANT_QOS_DEFAULT 489, 490
- DDS_PartitionQosPolicy 85
- DDS_PresentationQosPolicy 86
- DDS_PublicationMatchStatus 113
- DDS_Publisher_begin_coherent_changes . . . 247
- DDS_Publisher_copy_from_topic_qos 249
- DDS_Publisher_create_datawriter 250
- DDS_Publisher_delete_contained_entities . . 254
- DDS_Publisher_delete_datawriter 255
- DDS_Publisher_enable (inherited) 256
- DDS_Publisher_end_coherent_changes 256
- DDS_Publisher_get_default_datawriter_qos . 257
- DDS_Publisher_get_listener 259
- DDS_Publisher_get_participant 259
- DDS_Publisher_get_qos 260
- DDS_Publisher_get_status_changes (inherited) . . 261
- DDS_Publisher_get_statuscondition (inherited) . . 261
- DDS_Publisher_lookup_datawriter 261
- DDS_PUBLISHER_QOS_DEFAULT 492
- DDS_Publisher_resume_publications 262
- DDS_Publisher_set_default_datawriter_qos . 263
- DDS_Publisher_set_listener 265
- DDS_Publisher_set_qos 267
- DDS_Publisher_suspend_publications 269
- DDS_PublisherListener 30
- DDS_PublisherListener interface 335
- DDS_PublisherListener_alloc 338
- DDS_PublisherListener_on_liveliness_lost
(inherited, abstract) 338
- DDS_PublisherListener_on_offered_deadline_missed (inherited, abstract) 339
- DDS_PublisherListener_on_offered_incompatible_qos (inherited, abstract) 339
- DDS_PublisherListener_on_publication_match
(inherited, abstract) 339
- DDS_PublisherQos 491
- DDS_QueryCondition_get_datareader (inherited). 473
- DDS_QueryCondition_get_instance_state_mask
(inherited) 473
- DDS_QueryCondition_get_query_arguments . 473
- DDS_QueryCondition_get_query_expression . 474
- DDS_QueryCondition_get_sample_state_mask
(inherited) 475
- DDS_QueryCondition_get_trigger_value
(inherited) 475
- DDS_QueryCondition_get_view_state_mask
(inherited) 476
- DDS_QueryCondition_set_query_arguments . 476
- DDS_ReadCondition_get_datareader 468
- DDS_ReadCondition_get_instance_state_mask . . 469
- DDS_ReadCondition_get_sample_state_mask 470
- DDS_ReadCondition_get_trigger_value
(inherited) 470
- DDS_ReadCondition_get_view_state_mask . 471
- DDS_ReaderDataLifecycleQosPolicy 92
- DDS_ReliabilityQosPolicy 93
- DDS_RequestedDeadlineMissedStatus 114
- DDS_RequestedIncompatibleQosStatus 114
- DDS_ResourceLimitsQosPolicy 95
- DDS_SampleInfo 449
- DDS_SampleLostStatus 116
- DDS_SampleRejectedStatus 117
- DDS_sequence_get_release 17
- DDS_sequence_set_release 16
- DDS_StatusCondition_get_enabled_statuses . 132
- DDS_StatusCondition_get_entity 133
- DDS_StatusCondition_get_trigger_value
(inherited) 134
- DDS_StatusCondition_set_enabled_statuses . 134
- DDS_string_alloc 20
- DDS_Subscriber 545
- DDS_Subscriber Statecraft for a Read
Communication Status 546
- DDS_Subscriber_begin_access 350
- DDS_Subscriber_copy_from_topic_qos 350
- DDS_Subscriber_create_datareader 351
- DDS_Subscriber_delete_contained_entities . 355
- DDS_Subscriber_delete_datareader 356
- DDS_Subscriber_enable (inherited) 358
- DDS_Subscriber_end_access 358
- DDS_Subscriber_get_datareaders 358
- DDS_Subscriber_get_default_datareader_qos . 358
- DDS_Subscriber_get_listener 360
- DDS_Subscriber_get_participant 360
- DDS_Subscriber_get_qos 361
- DDS_Subscriber_get_status_changes (inherited) .

362	
DDS_Subscriber_get_statuscondition (inherited).	362
DDS_Subscriber_lookup_datareader	363
DDS_Subscriber_notify_datareaders	363
DDS_SUBSCRIBER_QOS_DEFAULT	493
DDS_Subscriber_set_default_datareader_qos	365
DDS_Subscriber_set_listener	366
DDS_Subscriber_set_qos	369
DDS_SubscriberListener	32
DDS_SubscriberListener__alloc	454
DDS_SubscriberListener_on_data_available (inherited, abstract)	455
DDS_SubscriberListener_on_data_on_readers (abstract)	455
DDS_SubscriberListener_on_liveliness_changed (inherited, abstract)	457
DDS_SubscriberListener_on_requested_deadline _missed (inherited, abstract)	457
DDS_SubscriberListener_on_requested_incompat ible_qos (inherited, abstract)	457
DDS_SubscriberListener_on_sample_lost (inherited, abstract)	457
DDS_SubscriberListener_on_sample_rejected (inherited, abstract)	458
DDS_SubscriberListener_on_subscription_match (inherited, abstract)	458
DDS_SubscriberQos	492
DDS_SubscriptionMatchStatus	118
DDS_TimeBasedFilterQosPolicy	98
DDS_Topic Definition Example	553
DDS_Topic_enable (inherited)	216
DDS_Topic_get_inconsistent_topic_status . .	216
DDS_Topic_get_listener	217
DDS_Topic_get_name (inherited)	218
DDS_Topic_get_participant (inherited)	218
DDS_Topic_get_qos	218
DDS_Topic_get_statuschanges (inherited) . .	219
DDS_Topic_get_statuscondition (inherited) .	220
DDS_Topic_get_type_name (inherited)	220
DDS_TOPIC_QOS_DEFAULT	495
DDS_Topic_set_listener	220
DDS_Topic_set_qos	222
DDS_TopicDataQosPolicy	98
DDS_TopicDescription_get_name	213
DDS_TopicDescription_get_participant	213
DDS_TopicDescription_get_type_name	214
DDS_TopicListener	29
DDS_TopicListener interface	235
DDS_TopicListener__alloc	236
DDS_TopicListener_on_inconsistent_topic (abstract)	236
DDS_TopicQos	493
DDS_TransportPriorityQosPolicy	99
DDS_TypeSupport__alloc (abstract)	239
DDS_TypeSupport_get_type_name (abstract)	239
DDS_TypeSupport_register_type (abstract) .	239
DDS_UserDataQosPolicy	100
DDS_WaitSet__alloc	119
DDS_WaitSet_attach_condition	120
DDS_WaitSet_detach_condition	121
DDS_WaitSet_get_conditions	123
DDS_WaitSet_wait	124
DDS_WriterDataLifecycleQosPolicy	100
Default attributes	59
Document Structure	3
Domain Module	39, 136

F

Functionality	37
-------------------------	----

I

IDL Mapping Rules for Sequences	9	Infrastructure Module	38, 48
IDL Pre-processor	554	Inheritance of Abstract Operations	35
IDL to Host Language Mapping	554	instance_state	531

L

Listeners546 Listeners Interfaces 22

M

Memory Management9

O

Operations.....4 Operations Concerning States 535

P

Plain Communication Status State Chart544 Data Type “Foo”212
 Plain Sequences10 Publication Module42, 244
 Pre-defined Bit Mask Definitions.....534 Publication Type Specific Classes272
 Pre-processor Generation of the Typed Classes for

Q

QosPolicy Basics62 QosPolicy Objects 11
 QosPolicy Default Attributes59 QosPolicy Settings56

R

read535 DDS_DestinationOrderQosPolicy67
 Read Communication Status sDDS_DataReader
 Statecraft545 Requested/Offered DDS_DurabilityQosPolicy .69
 read_instance537 Requested/Offered DDS_ReliabilityQosPolicy 83,
 read_next_sample.....536 95
 read_w_condition.....536 Resources and operations.....12
 Requested Offered PresentationQosPolicy.....91 Return Codes7
 Requested/Offered57 Return Value254
 Requested/Offered RxO61

S

sample_state529 Snapshot532, 533
 SampleInfo Class529 SPACE_FooDataReader_create_querycondition
 Sequences11 (inherited)415
 Sequences DDS_13 SPACE_FooDataReader_create_readcondition
 Sequences Embedded in QosPolicy Objects11 (inherited)415
 Sequences Embedded in Status Objects.....12 SPACE_FooDataReader_delete_contained_entitie
 Signal Handling8 s (inherited)415

- SPACE_FooDataReader_delete_readcondition
(inherited). 415
- SPACE_FooDataReader_enable (inherited) . . 416
- SPACE_FooDataReader_get_key_value . . . 416
- SPACE_FooDataReader_get_listener (inherited).
417
- SPACE_FooDataReader_get_liveliness_changed_
status (inherited). 417
- SPACE_FooDataReader_get_matched_publicatio
n_data (inherited). 418
- SPACE_FooDataReader_get_matched_publicatio
ns (inherited) 418
- SPACE_FooDataReader_get_qos (inherited). 418
- SPACE_FooDataReader_get_requested_deadline_
missed_status (inherited) 418
- SPACE_FooDataReader_get_requested_incompat
ible_qos_status (inherited). 419
- SPACE_FooDataReader_get_sample_lost_status
(inherited). 419
- SPACE_FooDataReader_get_sample_rejected_sta
tus (inherited). 419
- SPACE_FooDataReader_get_status_changes
(inherited). 419
- SPACE_FooDataReader_get_statuscondition
(inherited). 420
- SPACE_FooDataReader_get_subscriber
(inherited). 420
- SPACE_FooDataReader_get_subscription_match_
status (inherited). 420
- SPACE_FooDataReader_get_topicdescription
(inherited). 420
- SPACE_FooDataReader_read 421
- SPACE_FooDataReader_read_instance . . . 426
- SPACE_FooDataReader_read_next_instance 428
- SPACE_FooDataReader_read_next_instance_w_
condition. 431
- SPACE_FooDataReader_read_next_sample . 433
- SPACE_FooDataReader_read_w_condition . 433
- SPACE_FooDataReader_return_loan. 435
- SPACE_FooDataReader_set_listener (inherited).
438
- SPACE_FooDataReader_set_qos (inherited) . 438
- SPACE_FooDataReader_take. 438
- SPACE_FooDataReader_take_instance . . . 440
- SPACE_FooDataReader_take_next_sample . 446
- SPACE_FooDataReader_take_w_condition . 446
- SPACE_FooDataReader_wait_for_historical_data
(inherited). 448
- SPACE_FooDataWriter_assert_liveliness
(inherited). 304
- SPACE_FooDataWriter_dispose 304
- SPACE_FooDataWriter_dispose_w_timestamp .
308, 335
- SPACE_FooDataWriter_enable (inherited) . . 310
- SPACE_FooDataWriter_get_key_value. . . . 310
- SPACE_FooDataWriter_get_listener (inherited) .
311
- SPACE_FooDataWriter_get_liveliness_lost_statu
s (inherited) 312
- SPACE_FooDataWriter_get_matched_subscriptio
n_data (inherited). 312
- SPACE_FooDataWriter_get_matched_subscriptio
ns (inherited) 312
- SPACE_FooDataWriter_get_offered_deadline_mi
ssed_status (inherited) 312
- SPACE_FooDataWriter_get_offered_incompatibl
e_qos_status (inherited). 313
- SPACE_FooDataWriter_get_publication_match_s
tatus (inherited) 313
- SPACE_FooDataWriter_get_publisher (inherited)
313
- SPACE_FooDataWriter_get_qos (inherited) . 313
- SPACE_FooDataWriter_get_status_changes
(inherited). 314
- SPACE_FooDataWriter_get_statuscondition
(inherited). 314
- SPACE_FooDataWriter_get_topic (inherited) 314
- SPACE_FooDataWriter_register_instance. . 314
- SPACE_FooDataWriter_register_instance_w_tim
estamp 318
- SPACE_FooDataWriter_set_listener (inherited) .
319
- SPACE_FooDataWriter_set_qos (inherited) . 320
- SPACE_FooDataWriter_unregister_instance. 320
- SPACE_FooDataWriter_unregister_instance_w_ti
mestamp. 324
- SPACE_FooDataWriter_write 326
- SPACE_FooDataWriter_write_w_timestamp 330
- SPACE_FooTypeSupport__alloc 240
- SPACE_FooTypeSupport_get_type_name. . . 241

SPACE_FooTypeSupport_register_type	242	State Masks	534
SQL Examples	561	State per Sample.	530
SQL Grammar in BNF	559	Status Description Per DDS_Entity	103
SQL Token Expression.	560	Status Objects.	12
Standard Defined Type.	10	Status Per DDS_Entity	130
State Chart of the instance_state for a Single Instance	532	Struct DDS_Listener	24
State Chart of the sample_state for a Single Sample 530		Struct DDS_SampleInfo.	448
State Chart of the view_state for a Single Instance 533		Struct DDS_Status	102
State Definitions	534	Struct QosPolicy.	55

T

take	536	Thread Safety	8
take_next_instance	442	Topic-Definition Module	40, 211
take_next_sample	536	Topic-Definition type specific classes	237
take_w_condition	536	Trigger State of the DDS_GuardCondition	552
The DCPS Publication Module's Class Model	244	Trigger State of the DDS_ReadCondition and DDS_QueryCondition.	551
The DCPS Subscription Module's Class Model . . 347		Trigger State of the DDS_StatusCondition	551

U

User Defined Type	10
-----------------------------	----

V

view_state	532
----------------------	-----

