# RTI Connext Micro User's Manual

**Version 4.3.0**

# Contents


**1 Getting Started** **1**

1.1 Before You Get Started . . . 1
1.1.1 What is RTI Connext Micro? . . . 1
1.1.2 Downloading Connext Micro . . . 2
1.1.3 Installing Connext Micro . . . 2
Installing with a GUI . . . 3
Installing from a command line . . . 3
Paths mentioned in documentation . . . 4
1.1.4 Checking what is installed . . . 6
1.1.5 Where do I get more help? . . . 6
1.2 Hello World with Publish/Subscribe . . . 6
1.2.1 Introduction to DataWriters, DataReaders, and Topics . . . 7
1.2.2 Create a DataWriter and DataReader . . . 8
Set environment variables . . . 8
Generate an example application . . . 8
Modify application . . . 11
Modify publisher . . . 11
Modify subscriber . . . 13
Compile your changes . . . 14
Run the applications . . . 14
Taking it further . . . 16
1.2.3 Troubleshooting . . . 18
Why aren't my applications communicating? . . . 18
Why does the DataReader miss the first samples? . . . 18
1.2.4 Next steps . . . 18

**2 Developing Applications** **19**

2.1 Prepare Your Development Environment . . . 19
2.1.1 Set environment variables . . . 19
2.1.2 Add required preprocessor flags . . . 19
2.1.3 Link applications and libraries . . . 20
2.2 Define a Data Type . . . 21
2.3 Generate Type Support Code with rtiddsgen . . . 22
2.4 Create an Application . . . 23
2.4.1 Registry Configuration . . . 23
2.5 Configure UDP Transport . . . 25
2.6 Create DomainParticipant, Topic, and Type . . . 26

2.6.1 Register Type . . . 28
2.6.2 Create Topic of Registered Type . . . 28
2.6.3 DPSE Discovery: Assert Remote Participant . . . 28
2.7 Create Publisher . . . 29
2.8 Create DataWriter . . . 29
2.8.1 DPSE Discovery: Assert Remote Subscription . . . 30
2.8.2 Writing Samples . . . 31
2.9 Create Subscriber . . . 32
2.10 Create DataReader . . . 32
2.10.1 DPSE Discovery: Assert Remote Publication . . . 34
2.10.2 Receiving Samples . . . 35
2.10.3 Filtering Samples . . . 37

**3 Programmer's Guide 38**
3.1 Data Types . . . 38
3.1.1 Introduction to the Type System . . . 39
Sequences . . . 40
Strings and Wide Strings . . . 41
Extensible Types (XTypes) 1.2 Compatibility . . . 42
3.1.2 Creating User Data Types with IDL . . . 42
3.1.3 Working with DDS Data Samples . . . 43
3.2 DDS Entities . . . 43
3.3 Sending Data . . . 44
3.3.1 Preview: Steps to Sending Data . . . 45
3.3.2 Publishers . . . 45
3.3.3 DataWriters . . . 46
3.3.4 Supported QoS policies . . . 46
3.4 Receiving Data . . . 46
3.4.1 Preview: Steps to Receiving Data . . . 47
3.4.2 Subscribers . . . 49
3.4.3 DataReaders . . . 49
3.4.4 Using DataReaders to Access Data (Read & Take) . . . 49
3.4.5 Supported QoS policies . . . 50
3.5 DDS Domains . . . 50
3.5.1 Fundamentals of DDS Domains and DomainParticipants . . . 50
3.5.2 Discovery Announcements . . . 52
3.6 Discovery . . . 52
3.6.1 What is Discovery? . . . 53
Simple Participant Discovery . . . 53
Simple Endpoint Discovery . . . 54
3.6.2 Configuring Participant Discovery Peers . . . 55
The Peer Descriptor . . . 55
3.6.3 Configuring Initial Peers and Adding Peers . . . 56
3.6.4 Configuring Discovery Data Reception . . . 57
3.6.5 Configuring User Data Reception . . . 58
3.6.6 Configuring User Data Reception per DataReader or DataWriter . . . 60
3.6.7 Discovery Plugins . . . 60
Dynamic Discovery Plugin . . . 60

Static Discovery Plugin . . . 61
3.6.8 Asymmetric Matching and Lost Samples . . . 63
3.6.9 DomainParticipant Discovery by Name . . . 64
3.6.10 Queueing Discovery Messages . . . 64
3.6.11 Restarting Discovery . . . 65
How CPU suspension affects DDS discovery . . . 65
How to restart discovery . . . 66
Usage considerations . . . 68
3.6.12 Interoperating with Connext Professional Discovery . . . 68
3.7 User Discovery Data . . . 68
3.7.1 Introduction . . . 68
3.7.2 Resource Limits . . . 69
3.7.3 Propagating User Discovery Data . . . 70
3.7.4 Accessing User Discovery Data . . . 70
3.7.5 QoS Policies . . . 72
USER_DATA . . . 72
TOPIC_DATA . . . 73
GROUP_DATA . . . 73
3.8 Partitions . . . 73
3.8.1 Introduction . . . 73
3.8.2 Rules for partition matching . . . 75
3.8.3 Pattern matching for partition names . . . 75
Regular Expression Matching . . . 76
3.8.4 Example . . . 77
Location-based partitions . . . 78
Access-control group partitions . . . 79
3.8.5 Properties . . . 79
3.8.6 Resource limits . . . 79
Configuring for runtime allocation . . . 80
Configuring for preallocated memory . . . 80
3.9 Content Filtering . . . 81
3.9.1 Creating a content filter . . . 81
3.9.2 Where filtering is applied . . . 84
3.9.3 Interoperability . . . 86
With Connext Professional . . . 86
With Connext Cert . . . 86
3.9.4 SQL filter expression notation . . . 86
Example SQL filter expressions . . . 87
SQL grammar . . . 88
Token expressions . . . 89
Type compatibility in the predicate . . . 90
Enumerations . . . 91
3.9.5 Custom filters . . . 91
Registering a custom filter . . . 91
Unregistering a custom filter . . . 93
Implementing DDS_ContentFilterI . . . 94
Threading model . . . 95
Preallocating compiled filter resources . . . 95

3.10 Generating Type Support with rtiddsgen . . . 98
3.10.1 Why Use rtiddsgen? . . . 98
3.10.2 IDL Type Definition . . . 98
3.10.3 Generating Type Support . . . 98
Generating type support for C . . . 99
Generating type support for C++ . . . 99
Notes on command line options . . . 99
Generated type support files . . . 99
3.10.4 Using Custom Types in Connext Micro Applications . . . 100
3.10.5 Customizing Generated Code . . . 100
3.10.6 Unsupported Features of rtiddsgen with Connext Micro . . . 101
3.11 Threading Model . . . 101
3.11.1 Introduction . . . 101
3.11.2 Architectural Overview . . . 101
3.11.3 Threading Model . . . 102
OSAPI Threads . . . 102
UDP Transport Threads . . . 103
General Thread Configuration . . . 104
3.11.4 Thread Safety . . . 104
Calling DDS APIs from listeners and callbacks . . . 105
Calling DDS APIs from a type-plugin . . . 106
3.11.5 Critical Sections . . . 106
Calling DDS APIs from listeners . . . 106
3.12 Batching . . . 106
3.12.1 Overview . . . 107
3.12.2 Interoperability . . . 107
3.12.3 Performance . . . 107
3.12.4 Example Configuration . . . 107
3.13 Message Integrity Checking . . . 109
3.13.1 RTPS checksum . . . 109
3.13.2 Configurations . . . 110
Selecting a checksum algorithm . . . 110
Configuring the DDS DomainParticipant . . . 110
3.13.3 Participant discovery and compatibility . . . 111
3.13.4 Interoperability with Connext Professional . . . 112
3.14 Sending Large Data . . . 112
3.14.1 How Sending Large Data Works . . . 113
3.14.2 Key Properties of Fragmentation . . . 113
3.14.3 Differences Between Fragmentation and Transport Limits . . . 113
3.14.4 Components Involved in Sending Large Data . . . 114
3.14.5 Configuring Large Data Support . . . 114
Task scheduler configuration . . . 114
Flow controller configuration . . . 115
Writer resource limits . . . 115
Reader fragment reassembly limits . . . 116
3.14.6 When to Choose Synchronous vs Asynchronous Publishing . . . 116
3.14.7 Complete Example . . . 116
3.14.8 Limitations . . . 117

3.15 Zero Copy Transfer . . . 118
3.15.1 Compatibility . . . 119
3.15.2 Overview . . . 120
3.15.3 Getting started . . . 120
Writing samples . . . 121
Reading samples . . . 122
3.15.4 Synchronizing samples . . . 123
Zero Copy v1 synchronization . . . 123
Zero Copy v2 synchronization . . . 124
3.15.5 Caveats . . . 125
3.16 FlatData Language Binding . . . 125
3.16.1 Overview . . . 125
3.16.2 Getting Started . . . 125
3.16.3 Further Information . . . 126
3.17 Application Generation Using XML . . . 126
3.17.1 Defining an Application in XML . . . 127
Important Points . . . 127
3.17.2 Generating the Application from XML . . . 128
Micro Application Generator (MAG) Tool . . . 128
Generating the Application with MAG . . . 129
MAG Command-Line Options . . . 130
Integrating Generated Files into Your Application's Build . . . 134
3.17.3 Creating the Application . . . 134
Generate Type-Support Code from the Type Definition . . . 134
Generate DDS Entities from the System Definition . . . 136
Examining the application . . . 136
Call API to Create DomainParticipant . . . 137
Retrieve Entities by Name . . . 137
3.17.4 Examine example XML configuration files . . . 137
Type Definition . . . 138
Domain Definition . . . 139
DomainParticipant Definition . . . 140
QoS Definition . . . 145
Transport and Discovery Configuration . . . 158
Flow Controllers . . . 167
Static Discovery . . . 170
Lightweight Security Plugin . . . 177
Content filtering . . . 180
3.17.5 Generating AUTOSAR Applications . . . 187
Define your system in DDS system XML . . . 188
Run MAG with deployment and application type selectors . . . 194
Generate AUTOSAR type ARXML with rtiarcgen . . . 195
Import both ARXML outputs into AUTOSAR toolchain . . . 197
Examples of generated code . . . 198
Custom AUTOSAR Templates . . . 207
Limitations and considerations . . . 208
3.17.6 Generating native DDS applications with deployment-aware code generation 209
Define your system in DDS system XML . . . 209

Run MAG with deployment and application type selectors . . . 210
Generate type support files with rtiddsgen . . . 211
Understanding the generated files . . . 212
Building and running native applications . . . 213
Customizing native applications . . . 214
Troubleshooting . . . 214
3.17.7 Errors caused by invalid configurations . . . 215
3.18 QoS Mapping in XML . . . 217
3.18.1 Domain Participant Qos QoS Mapping . . . 218
discovery_config . . . 218
default_unicast . . . 219
resource_limits . . . 219
3.18.2 Domain Participant Factory Qos QoS Mapping . . . 219
Top-level fields . . . 220
Nested resource_limits (systemResourceLimitsQosPolicy) . . . 220
3.18.3 Publisher QoS mapping . . . 220
Top-level fields . . . 221
3.18.4 Subscriber QoS mapping . . . 221
Top-level fields . . . 222
3.18.5 Topic QoS mapping . . . 222
Top-level fields . . . 223
3.18.6 DataReader QoS mapping . . . 224
Top-level fields . . . 225
3.18.7 DataWriter QoS mapping . . . 226
Top-level fields . . . 227
3.18.8 Structural elements mapping . . . 228
`filter` and `content_filter` (DataReader element) . . . 228
`register_type` (Domain and Participant element) . . . 229
`multiplicity` (Publisher, Subscriber, DataWriter, DataReader attribute) . . . 229
base_name and base_names (QoS profile inheritance) . . . 230
topic (Domain element) . . . 230
data_writer (Publisher, Participant element) . . . 231
data_reader (Subscriber, Participant element) . . . 231
publisher (Participant element) . . . 232
subscriber (Participant element) . . . 233
domain_participant (ParticipantLibrary element) . . . 233
domain (DomainLibrary element) . . . 235
domain_library and participant_library (Root elements) . . . 235
3.19 Building Against FACE Conformance Libraries . . . 236
3.19.1 Requirements . . . 236
Connext Micro source code . . . 236
FACE Conformance Tools . . . 236
CMake . . . 236
3.19.2 FACE golden libraries . . . 236
Building the FACE golden libraries . . . 236
3.19.3 Building instructions . . . 237
3.20 Working With Sequences . . . 239
3.20.1 Introduction . . . 239

3.20.2 Working with Sequences . . . 239
Overview . . . 239
Working with IDL sequences . . . 241
Working with application defined sequences . . . 242
3.21 Example Applications . . . 244
3.21.1 Generating examples . . . 244
Default example . . . 244
Custom example . . . 244
Descriptions of generated examples . . . 245
How to compile and run the generated examples . . . 247
3.21.2 Provided examples . . . 248
3.22 Lightweight Security Plugin . . . 248
3.22.1 Overview . . . 248
3.22.2 Downloading and Installing the Lightweight Security Plugin . . . 249
Package contents . . . 249
3.22.3 Getting Started . . . 249
Include header files . . . 250
Link application and libraries . . . 250
Enable lightweight security in your application . . . 250
Specify the plugin in the DomainParticipant QoS . . . 251
Configure the plugin . . . 252
Create a protected DomainParticipant . . . 253
3.22.4 Configuring the Lightweight Security Plugin . . . 253
3.22.5 Technical Considerations . . . 255
RTPS transport only . . . 255
Maximum Transmission Unit (MTU) . . . 255
3.22.6 Interoperability . . . 255
Additional Authenticated Data (AAD) . . . 255
Passphrase ID limitations . . . 256
3.23 Memory Management . . . 256
3.23.1 Resource limits . . . 256
3.23.2 Dynamic memory allocation . . . 256
3.23.3 DDS resource limits . . . 257
3.23.4 Discovery plugin resource limits . . . 257
3.23.5 Type support resource limits . . . 258
3.24 Transport Priority . . . 259
3.24.1 Using transport priority with UDP . . . 259
3.24.2 Configuring transport priority . . . 260
3.24.3 Examples . . . 262
3.24.4 Enforcing priority selection . . . 262
3.24.5 System resource and performance considerations . . . 262
3.25 Startup Time Guidelines . . . 263
3.25.1 Understanding the startup process . . . 263
Initialization time . . . 263
Discovery time . . . 263
3.25.2 Minimizing startup time . . . 264
Application design . . . 264
QoS and resource configuration . . . 265

Build settings . . . 268
3.25.3 Measuring startup time . . . 268
How to measure initialization time . . . 268
How to measure discovery time . . . 268
What to record . . . 269
3.25.4 Validating startup time improvements . . . 269

**4 Transports** **270**
4.1 Introduction . . . 270
4.2 Transport Registration . . . 271
4.3 Transport Addresses . . . 272
4.3.1 Reserving addresses and ports . . . 272
4.3.2 Address limitations . . . 273
4.3.3 Address notation . . . 273
4.4 Transport Port Number . . . 274
4.5 Interoperating with Connext Professional Transports . . . 274
4.6 RTPS . . . 274
4.6.1 Registration of RTPS . . . 274
4.6.2 Overriding the Builtin RTPS Checksum Functions . . . 276
Checksum function definition . . . 276
4.6.3 Example . . . 277
4.7 INTRA Transport . . . 279
4.7.1 Registering the INTRA Transport . . . 279
4.7.2 Configuring QoS policies . . . 280
Reliability . . . 280
Durability . . . 280
4.7.3 Threading Model . . . 280
4.8 Shared Memory Transport (SHMEM) . . . 281
4.8.1 Registering the SHMEM Transport . . . 281
4.8.2 Threading Model . . . 283
Configuring SHMEM Receive Threads . . . 283
4.8.3 SHMEM Configuration . . . 284
received_message_count_max . . . 284
receive_buffer_size . . . 284
message_size_max . . . 285
recv_thread_property . . . 285
pro_minimum_compatiblity_version . . . 285
4.8.4 Caveats . . . 285
Leftover shared memory resources . . . 285
Darwin and Linux systems . . . 285
QNX systems . . . 286
Windows and VxWorks systems . . . 286
4.9 Zero Copy v2 Transport . . . 286
4.9.1 Generate example and type support files . . . 286
4.9.2 Initialize the Zero Copy v2 transport . . . 287
4.9.3 Register the Zero Copy v2 transport . . . 287
4.9.4 Enable transports . . . 288
4.9.5 Sample management . . . 289

4.9.6 Limit type plugin heap creation . . . 290
4.9.7 Configuration . . . 291
4.9.8 Using multiple Zero Copy v2 transport instances . . . 292
4.10 UDP Transport . . . 293
4.10.1 Registering the UDP Transport . . . 293
4.10.2 Threading Model . . . 294
Configuring UDP Receive Threads . . . 295
4.10.3 UDP Configuration . . . 296
allow_interface . . . 296
deny_interface . . . 296
max_send_buffer_size . . . 296
max_receive_buffer_size . . . 296
max_message_size . . . 296
multicast_ttl . . . 297
nat . . . 297
if_table . . . 298
multicast_interface . . . 298
is_default_interface . . . 298
disable_auto_interface_config . . . 299
multicast_loopback_disabled . . . 299
recv_thread . . . 299
enable_interface_bind . . . 299
disable_multicast_bind . . . 299
disable_multicast_interface_select . . . 299
source_rules . . . 300
destination_rules . . . 300
transform_udp_mode . . . 300
transform_locator_kind . . . 300
4.10.4 UDP Transformations . . . 300
Overview . . . 301
Enabling UDP Transformation . . . 302
Creating a Transformation Library . . . 303
Creating Transformation Rules . . . 304
Interoperability . . . 304
Error Handling . . . 304
Example Code . . . 304
Examples . . . 324
4.11 NETIO Datagram Transport . . . 330
4.11.1 Registering a Datagram Interface . . . 330
4.11.2 Addressing a Datagram Transport . . . 331
4.11.3 Datagram UDP Setup . . . 332
Datagram Shared flag . . . 334
4.11.4 User Interface . . . 334

**5 Debugging** **336**
5.1 Overview . . . 336
5.2 Configuring Logging . . . 336
5.3 Log Message Kinds . . . 337

5.4 Interpreting Log Messages and Error Codes . . . 337
5.5 Writing a Custom Log Handler . . . 338
5.6 Contact Support . . . 341
5.7 Join the Community . . . 342

**6 Platforms Guide 343**
6.1 Introduction . . . 343
6.1.1 Library types . . . 343
Integrated libraries . . . 344
Split libraries . . . 344
6.1.2 Library descriptions . . . 345
6.1.3 Build profiles . . . 346
6.1.4 Supported libraries by platform . . . 347
6.1.5 Supported transports by platform . . . 351
6.2 Building Connext Micro for Common Platforms . . . 353
6.2.1 Setting up the build environment . . . 353
The host environment . . . 353
The target environment . . . 354
6.2.2 Building the PSL . . . 354
Building the PSL with rtime-make . . . 355
Building the PSL with CMake . . . 356
6.2.3 Building the source . . . 358
Building with rtime-make . . . 358
Building with CMake . . . 359
6.2.4 Cross-compiling Connext Micro . . . 362
6.3 Building Connext Micro with Compatibility for Connext Cert . . . 363
6.3.1 Compiling with compatibility for Connext Cert . . . 364
6.3.2 Compiling applications with compatibility for Connext Cert . . . 364
6.4 GUID Generation . . . 365
6.4.1 GUID format . . . 365
6.4.2 General guidelines . . . 365
6.4.3 Testing . . . 366
6.4.4 Example . . . 366
6.4.5 Implementing a custom GUID generation function . . . 367
6.5 FreeRTOS Platforms . . . 368
6.5.1 Port overview . . . 368
6.5.2 How to configure lwIP and FreeRTOS . . . 369
6.5.3 How the PIL was built for FreeRTOS . . . 380
6.5.4 Building the PSL from source for FreeRTOS platforms . . . 382
6.5.5 Building FreeRTOS applications with Connext Micro . . . 383
6.5.6 System tick rollovers . . . 385
6.6 Linux Platforms . . . 386
6.6.1 How the PIL was built for Linux platforms . . . 386
6.6.2 Building the PSL from source for Linux platforms . . . 388
6.7 MicroSAR Platforms . . . 390
6.7.1 Port overview . . . 390
6.7.2 How to configure MicroSAR . . . 390
DDS CDD configuration . . . 390

BSW configuration . . . 392
DomainParticipant configuration . . . 394
OSAPI_PortProperty reference . . . 394
6.7.3 NoCom/FullCom socket recovery . . . 405
API . . . 406
6.7.4 How the PIL was built for MicroSAR platforms . . . 406
6.7.5 Building the PSL from source for MicroSAR platforms . . . 409
6.7.6 Known issues and limitations . . . 411
6.7.7 Example . . . 411
6.8 macOS Platforms . . . 412
6.8.1 How the PIL was built for macOS platforms . . . 412
6.8.2 Building the PSL from source for macOS platforms . . . 413
6.9 QNX Platforms . . . 415
6.9.1 How the PIL was built for QNX platforms . . . 415
6.9.2 Building the PSL from source for QNX platforms . . . 417
6.10 Windows Platforms . . . 420
6.10.1 How the PIL was built for Windows platforms . . . 420
6.10.2 Building the PSL from source for Windows platforms . . . 421

**7 Connext Tools 423**
7.1 Admin Console . . . 423
7.2 Distributed Logger . . . 424
7.3 LabVIEW . . . 424
7.4 Monitor . . . 424
7.5 Recording Service . . . 424
7.5.1 RTI Recorder . . . 424
7.5.2 RTI Replay . . . 425
7.5.3 RTI Converter . . . 425
7.6 Wireshark . . . 425
7.7 Persistence Service . . . 425
7.8 Application Generation Using XML . . . 425

**8 Migrating from Connext Professional 426**
8.1 Supported DDS Features . . . 426
8.2 Development Environment . . . 426
8.3 Non-standard APIs . . . 427
8.4 QoS Policies . . . 427
8.5 Standard APIs . . . 427
8.6 IDL Files . . . 427

**9 API Reference 428**

**10 Release Notes 429**
10.1 Compatibility . . . 429
10.2 Supported Platforms and Programming Languages . . . 429
10.3 What's New in 4.3.0 . . . 431
10.3.1 Build Connext Micro applications for MicroSAR platforms . . . 431
10.3.2 Generate AUTOSAR code with Micro Application Generator . . . 431

DDS System XML model support . . . 432
Deployment-targeted code generation . . . 432
Application type selection . . . 432
10.3.3 Speed up initialization on AUTOSAR systems with configurable thread safety 432
10.3.4 Generate native applications with deployment-aware code generation . . . 432
10.3.5 Content filtering no longer experimental . . . 433
10.3.6 Stabilized builtin DDS SQL Content Filter configuration . . . 433
10.3.7 Expanded SQL content filter capabilities and improved robustness . . . 433
10.3.8 Reduce network and CPU usage with custom content filters . . . 433
10.3.9 Restart discovery after CPU suspension . . . 433
10.3.10 Prioritize data with the TRANSPORT_PRIORITY QoS policy . . . 433
10.3.11 Integrate rotating or on-demand credentials with Lightweight Security Plugin 434
10.3.12 Reduced heap memory usage . . . 434
10.3.13 Resume DDS traffic without restarting ECUs with socket recovery . . . 435
10.3.14 Default UDP transport name changed . . . 435
10.3.15 Streamlined default MAG messages . . . 435
10.3.16 Reduced runtime memory usage of MAG-generated code . . . 435
10.3.17 Third-party and open source software changes . . . 435
10.4 What's Fixed in 4.3.0 . . . 436
10.4.1 Discovery . . . 436
[Major] DomainParticipants did not use initial announcement configuration correctly . . . 436
[Minor] Periodic participant announcements not sent at expected rate . . . 436
[Trivial] Possible error when DPSE and participant_discovery_by_name are enabled . . . 436
10.4.2 Usability . . . 436
[Major] Network interfaces could not have non-ASCII characters in their name 436
10.4.3 Transports . . . 437
[Major] Zero Copy v1 limited the total number of shared memory segments to 500 . . . 437
[Trivial] UDP registration documented incorrectly . . . 437
[Trivial] Internal UDP pings failed to send . . . 437
10.4.4 Reliability Protocol and Wire Representation . . . 437
[Minor] DataWriter resent already-acknowledged samples when responding to ACKNACK on a lossy network . . . 437
10.4.5 Callbacks and Waitsets . . . 437
[Major] on_before callbacks could cause DataReader to stop receiving data . 437
[Minor] DataWriter did not invoke on_offered_incompatible_qos listener when data representations were incompatible . . . 438
10.4.6 Security Plugins SDK . . . 438
[Minor] Incorrect block count calculation for key rotation . . . 438
[Minor] OpenSSL PSK PSL implementation had NULL function pointers with no safeguards . . . 438
10.4.7 Content Filters and Query Conditions . . . 438
[Major] Possible failure to deliver samples when prior samples were filtered . 438
10.4.8 Logging . . . 438
[Trivial] MAG logged Java version warning on Windows systems . . . 438
10.4.9 APIs (C or Traditional C++) . . . 439

[Major] Possible stack buffer overflow due to DGRAM interface name length 439
[Major] Setting large max sequence size would produce integer overflow . . . 439
[Major] DataWriter's _w_timestamp operations did not detect non-normalized timestamp inputs . . . 439
[Major] Undefined DDS_TIME_INVALID variables . . . 439
[Minor] Misspelled OSAPI_LogProperty_INIITALIZER . . . 439
[Minor] DDS_Time_is_zero was not performing the correct evaluation . . . 440
[Minor] Low resolution timestamps could cause Zero Copy samples to remain visible longer than expected . . . 440
[Minor] Initializing discovery property with NULL value incorrectly returned PRECONDITION_NOT_MET . . . 440
[Minor] DDS_Entity_get_entity_kind was declared and documented but not defined . . . 440
[Trivial] LatencyBudget was missing from the API documentation . . . 440
[Trivial] False positive error messages when reattaching a condition to a WaitSet 440
10.4.10 Generated Code (C, Traditional C++, and Modern C++) . . . 441
[Major] Type registration API did not consistently report errors . . . 441
[Minor] Build error when generating an IDL that contained a long double on unsupported systems . . . 441
[Minor] Superflous parent member generated in a derived C++ class when using -interpreted 0 . . . 441
[Minor] Generated AUTOSAR files placed in the wrong folder for AUTOSAR_CDD . . . 441
[Trivial] Code Generator did not find Java when run from default directory . 441
10.4.11 Generated Code (Multiple Languages) . . . 442
[Major] Invalid generated code when type was inside modules with duplicate names . . . 442
10.4.12 OMG Specification Compliance . . . 442
[Minor] DDS_DURATION_INFINITE not compliant with OMG DDS specification . . . 442
10.4.13 Instances . . . 442
[Major] Possible failure to send dispose/unregister messages when using async publication . . . 442
[Minor] Instance state transition inconsistent with Connext Professional . . . 442
10.4.14 Crashes . . . 443
[Major] Content-filtered DataWriter could crash or fail to send messages . . . 443
[Major] Potential crash from malformed data fragments . . . 443
[Minor] XML parsing errors with special characters in comments . . . 443
10.4.15 Hangs . . . 443
[Critical] Potential deadlock when deleting a DomainParticipant that used the Flow Controller . . . 443
[Critical] Asynchronous DataWriter could silently stop sending samples and hang on deletion . . . 443
[Major] DataWriters stalled indefinitely when sharing flow controller . . . 444
10.4.16 Memory Leaks/Growth . . . 444
[Major] Possible memory leaks in CDR_WstringSeq_set_maximum . . . 444
[Major] Possible memory leaks in CDR_WstringSeq_copy . . . 444

[Major] DDS_DomainParticipantFactory_get_instance() not memory safe on some architectures . . . 444
[Major] Possible memory leak from failed write of a new instance . . . 444
[Minor] Possible memory leak when finalizing Publishers or Subscribers with Partition QoS . . . 445
[Minor] Possible memory leak when Entities experienced errors on topic creation 445
10.4.17 Data Corruption . . . 445
[Minor] Creating new DomainParticipant while receiving data could cause undefined behavior . . . 445
[Minor] DDS_WstringSeq instance could be returned in an inconsistent state when set_maximum failed . . . 445
10.4.18 Platform and Build Changes . . . 445
[Major] Possible transmission failure when SocketOwner was configured . . . 445
[Trivial] Not possible to disable thread semaphores . . . 446
[Trivial] rtime-make would not compile Lightweight Security Plugin outside of source directory . . . 446
[Trivial] Excluing DDS flow control and fragmentation resulted in compile time error . . . 446
10.4.19 Shipped Examples . . . 446
[Minor] Cert example template used undesirable code constructs . . . 446
[Trivial] The C++ example would attempt to delete invalid memory on specific failure conditions . . . 446
10.4.20 Interoperability . . . 446
[Critical] Failure to ignore vendor-specific PIDS . . . 446
[Critical] NULL pointer access in AUTOSAR Classic when a ping message received from Connext Professional 7.3.0 or higher . . . 447
[Major] KeyFlag for fragmented data interpreted incorrectly . . . 447
[Major] Samples of mutable types may have been dropped when bit FLAG_MUST_UNDERSTAND was set on key members . . . 447
10.4.21 Vulnerabilities . . . 448
[Critical] Potential stack buffer underflow in Connext Micro during message processing . . . 448
[Critical] Potential out-of-bounds read in Connext Micro during message processing . . . 448
[Minor] Potential-man-in-the-middle attack in Micro Application Generator when using Socket Appender in Apache Log4j™ . . . 449
10.4.22 Other . . . 449
[Critical] No communication on AUTOSAR systems with multiple IP addresses 449
[Major] Corrected ARXML reference generation for basic types . . . 450
[Major] DataReader could stop receiving samples due to resource miscount . 450
[Minor] Participant GUIDs generated on big-endian platforms may have reduced entropy . . . 450
[Minor] Errors in checksum documentation . . . 450
[Minor] Incorrect max size calculation for primitive array with CDR_LONG_DOUBLE_TYPE . . . 450
[Minor] DataWriter silently stopped sending data when flow controller token budget was smaller thanfragment size . . . 451
[Trivial] Possible warning during DomainParticipant shutdown . . . 451

[Trivial] rtime-make and rtiddsgen script errors when running from a directory with a space in its name . . . 451
[Trivial] Formatting errors in help printout for rtime-make.bat . . . 451
10.5 Previous Releases . . . 451
10.5.1 What's New in 4.2.0 . . . 451
Reduced core library size . . . 452
Secure RTPS communication with the Lightweight Security Plugin . . . 452
Optimize network bandwidth usage with content filtering . . . 452
Improve system robustness with DomainParticipant rediscovery . . . 452
Develop secure applications in XML with MAG . . . 453
Develop applications with content filtering in XML with MAG . . . 453
Enable DomainParticipant rediscovery in XML with MAG . . . 453
Enable endpoint discovery message queueing in XML with MAG . . . 453
Third-party and open source software changes . . . 453
10.5.2 What's Fixed in 4.2.0 . . . 454
Discovery . . . 454
Reliability Protocol and Wire Representation . . . 455
APIs (C or Traditional C++) . . . 456
Generated Code (C, Traditional C++, and Modern C++) . . . 456
Hangs . . . 457
Memory Leaks/Growth . . . 457
Platform and Build Changes . . . 457
Shipped Examples . . . 458
Interoperability . . . 458
Other . . . 458
10.5.3 What's New in 4.1.0 . . . 460
Platform-independent code is now separate from OS and network stack integration . . . 460
Transfer large data samples quickly with Zero Copy v2 . . . 461
Enable and configure Zero Copy transfer with MAG . . . 461
Enhance data reliability by detecting and discarding corrupted RTPS messages 462
Develop more reliable applications with MAG . . . 462
Guarantee compatibility with Connext Professional with MAG when using the Shared Memory Transport . . . 462
Improve control of data distribution to multicast addresses with new UDP transport options . . . 463
Develop applications with new UDP transport options with MAG . . . 463
Build Connext Micro libraries conveniently with symlinks . . . 463
10.5.4 What's Fixed in 4.1.0 . . . 464
Discovery . . . 464
Usability . . . 464
Transports . . . 465
Reliability Protocol and Wire Representation . . . 465
APIs (C or Traditional C++) . . . 466
XML Configuration . . . 467
Crashes . . . 467
Hangs . . . 468
Memory Leaks/Growth . . . 468

Data Corruption . . . 469
Interoperability . . . 469
Other . . . 469
10.5.5 What's New in 4.0.1 . . . 471
Enable or disable padding bits with PROPERTY QoS policy in DomainParticipant and Data Writer . . . 471
Generate examples with new template options for Code Generator . . . 472
10.5.6 What's Fixed in 4.0.1 . . . 472
Discovery . . . 472
Usability . . . 473
APIs (C or Traditional C++) . . . 473
Generated Code (C, Traditional C++, and Modern C++) . . . 474
Crashes . . . 475
Data Corruption . . . 475
Interoperability . . . 477
10.5.7 What's New in 4.0.0 . . . 477
Enhanced performance for asynchronous DataWriters . . . 477
Further control which entities communicate with each other using new Partition QoS policy . . . 478
Store additional entity-related information that is passed between applications during discovery using new User/Topic/Group Data QoS policies 478
Verify that locally created participant GUIDs are unique within a DomainParticipantFactory . . . 479
Micro Application Generator (MAG) . . . 479
10.5.8 What's Fixed in 4.0.0 . . . 481
Discovery . . . 482
Serialization and Deserialization . . . 482
Usability . . . 485
Transports . . . 485
Reliability Protocol and Wire Representation . . . 486
Logging . . . 486
Performance and Scalability . . . 487
APIs (C or Traditional C++) . . . 487
Generated Code (C, Traditional C++, and Modern C++) . . . 488
Micro Application Generator . . . 489
OMG Specification Compliance . . . 491
Interoperability . . . 491
Vulnerabilities . . . 492
Other . . . 493
10.6 Known Issues . . . 494
10.6.1 Possible memory leak when entity creation functions fail . . . 494
10.6.2 DomainParticipant_assert_liveliness does not function properly if Reader/Writer are in same DomainParticipant . . . 494
10.6.3 GCC warnings caused by type plugin functions . . . 494
10.6.4 MAG always generates initial value for AUTOSAR data type with one element 495
10.6.5 MAG does not support hex values as XML input . . . 495
10.6.6 Samples cannot be recovered if subscribing application fails to return loan . 495

10.6.7 64-bit discriminator values greater than (2^31-1) or smaller than (-2^31) not supported . . . 495
10.6.8 NaN and INF float and doubles are not detected and will not cause errors . 496
10.6.9 Ungracefully terminated QNX processes using SHMEM transport prevents startup of new processes due to unclosed POSIX semaphores . . . 496
10.6.10 Flow Controllers require RTOS . . . 496
10.6.11 Porting Guide not included . . . 496
10.6.12 Platform Independent Library toolchain dependencies . . . 497
10.6.13 Technical limitations for semaphore timeout variance on Windows systems . 497
10.7 Experimental Features . . . 497

**11 Benchmarks** **498**

**12 Copyrights** **499**

**13 Third-Party and Open Source Software** **501**
13.1 Connext Micro Libraries . . . 501
13.1.1 fnmatch . . . 501
13.1.2 crc32c.c . . . 502
13.1.3 MD5 . . . 503
13.2 RTI Code Generator (rtiddsgen) . . . 503
13.2.1 ANTLR . . . 503
13.2.2 ANTLR Runtime API . . . 504
13.2.3 Apache Commons CLI . . . 504
13.2.4 Apache Commons Lang . . . 504
13.2.5 Apache Commons Lang 3 . . . 504
13.2.6 Apache Log4j 2 . . . 504
13.2.7 Apache Velocity . . . 505
13.2.8 Simple Logging Facade for Java (SLF4J) . . . 505
13.2.9 Gson . . . 505
13.3 Micro Application Generator (rtiddsmag) . . . 506
13.3.1 Apache Commons CLI . . . 506
13.3.2 Apache Commons Lang 3 . . . 506
13.3.3 Apache Log4j 2 . . . 506
13.3.4 Apache Velocity . . . 506
13.3.5 Istack Common Utility Code Runtime . . . 506
13.3.6 JavaBeans Activation Framework API . . . 507
13.3.7 Javax Annotation API . . . 507
13.3.8 JAXB API . . . 507
13.3.9 JAXB Runtime . . . 507
13.3.10 Simple Logging Facade for Java (SLF4J) . . . 507
13.3.11 TXW2 . . . 508
13.4 Lightweight Security Plugin . . . 508
13.4.1 OpenSSL . . . 508
13.5 Licenses . . . 508
13.5.1 ANTLR 3 License . . . 508
13.5.2 Apache License version 2.0, January 2004 (http://www.apache.org/licenses/) 509
13.5.3 Common Development and Distribution License (CDDL) Version 1.1 . . . 513

**14 Appendices** **521**
14.1 Appendix A: AUTOSAR Integration Example . . . 521
14.1.1 How DDS integrates with AUTOSAR . . . 521
14.1.2 Integration steps . . . 523
Reference implementation . . . 523
Configuring the AUTOSAR BSW . . . 524
Configuring RTE . . . 528
Integrating source code . . . 530
Building and flashing . . . 532
Validating integration . . . 532
14.1.3 Understanding the architecture . . . 533
14.1.4 Customizing for your application . . . 534
Add a new data type and Topic . . . 534
Change QoS from Reliable to Best-Effort . . . 536
Tune timing for your application . . . 537
14.1.5 Troubleshooting . . . 537
DDS entities not created . . . 537
No DDS discovery . . . 538
Data not received . . . 538
High CPU usage . . . 538
Out of memory . . . 539
14.1.6 API quick reference . . . 539
14.1.7 Memory sizing . . . 540
14.1.8 Related documentation . . . 540
14.2 Appendix B: AUTOSAR Architecture Example . . . 540
14.2.1 System architecture . . . 540
14.2.2 RTE template layer . . . 541
Layer 2: Adapter layer . . . 543
Layer 3: DDS implementation layer . . . 546
14.2.3 Initialization sequence . . . 546
Ordering constraint . . . 547
14.2.4 Resource management . . . 548
Memory architecture . . . 548
AUTOSAR resource synchronization . . . 548
TcpIp socket management . . . 548
14.2.5 Timing architecture . . . 549
Runnable timing configuration . . . 549
14.2.6 Error handling strategy . . . 550
14.2.7 Scalability considerations . . . 551
14.3 Appendix C: Package Contents . . . 551
14.3.1 Host bundle . . . 551
14.3.2 Target bundle . . . 552
14.3.3 Lightweight Security Plugin host bundle . . . 553
14.3.4 Lightweight Security Plugin target bundle . . . 553
14.4 Appendix D: Directory Structure . . . 554

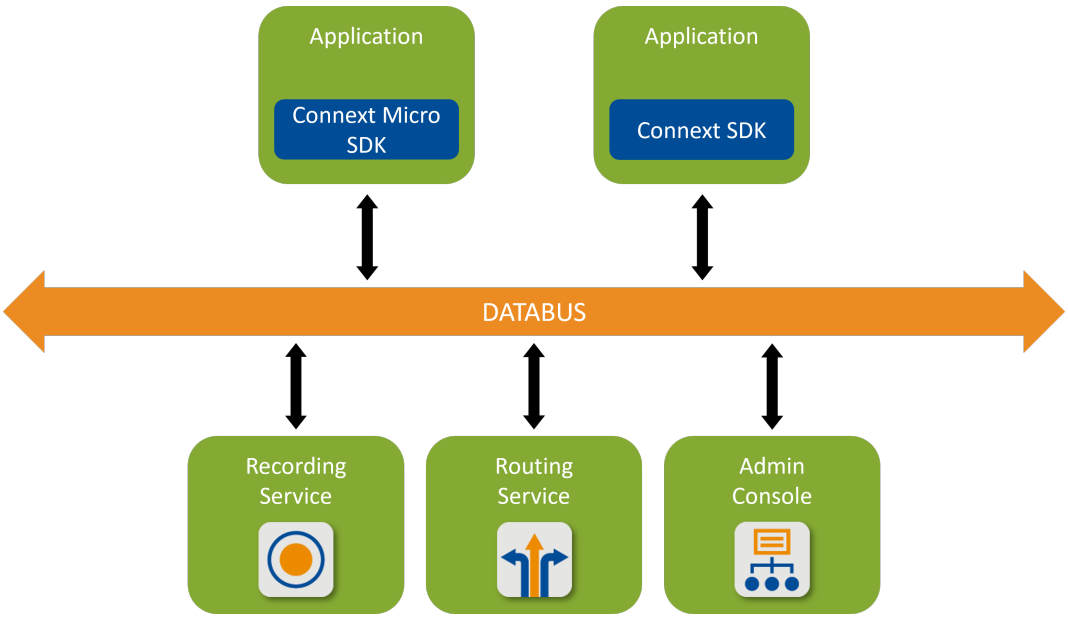
# Chapter 1

# Getting Started

Welcome to *RTI® Connext® Micro*! This section is for first-time users, and will walk you through the basics of *Connext Micro*. We will cover what *Connext Micro* is, how to download and install it, and how to run your first "Hello world!" application.

## 1.1 Before You Get Started

### 1.1.1 What is RTI Connext Micro?




*RTI® Connext® Micro®* is part of the *RTI® Connext®* connectivity framework for building distributed applications using a shared databus. *Connext Micro* provides C and C++ libraries and tools for resource-constrained embedded systems (that is, systems with limited memory and CPU power) to connect to the *Connext* databus, allowing them to exchange data without compromising performance or reliability. It interoperates seamlessly with other *Connext* applications, tools, and infrastructure services (with some exceptions as noted in this documentation).

RTI also offers *Connext Cert*, which provides C libraries and tools for safety-critical systems. *Connext Micro* includes C libraries that are comparable to *Connext Cert*'s feature set; you can use these to develop applications that can be migrated to *Connext Cert* with minimal changes. We describe this in detail in *Building Connext Micro with Compatibility for Connext Cert*. For more information on *Connext Cert*, please contact RTI.

**Warning:** Note that *Connext Micro* is **not** safety certifiable. The *Connext Cert*-comparable C libraries included with *Connext Micro* are intended for development purposes **only**.

### 1.1.2 Downloading Connext Micro

Once you have purchased *Connext Micro*, you can download the package files from support.rti.com. *Connext Micro* is provided in the following RTI package files (`.rtipkg`):

- `rti_connext_dds_micro-<version>-host.rtipkg`
- `rti_connext_dds_micro-<version>-target-<architecture>.rtipkg`

Download the host package with `<version>` 4.3.0, and a target package with `<version>` 4.3.0 and `<architecture>` matching your CPU and compiler. If you are unsure which `<architecture>` package to download for your platform, refer to the *Supported Platforms and Programming Languages* section and consult the **RTI Architecture Abbreviations** column in the Supported Platforms (PIL) table.

**Tip:** On the **Downloads** page, the **Info** button lists which versions of *Connext Professional* and *Connext Drive* are compatible with your *Connext Micro* release. See also the *Compatibility* section.

### 1.1.3 Installing Connext Micro

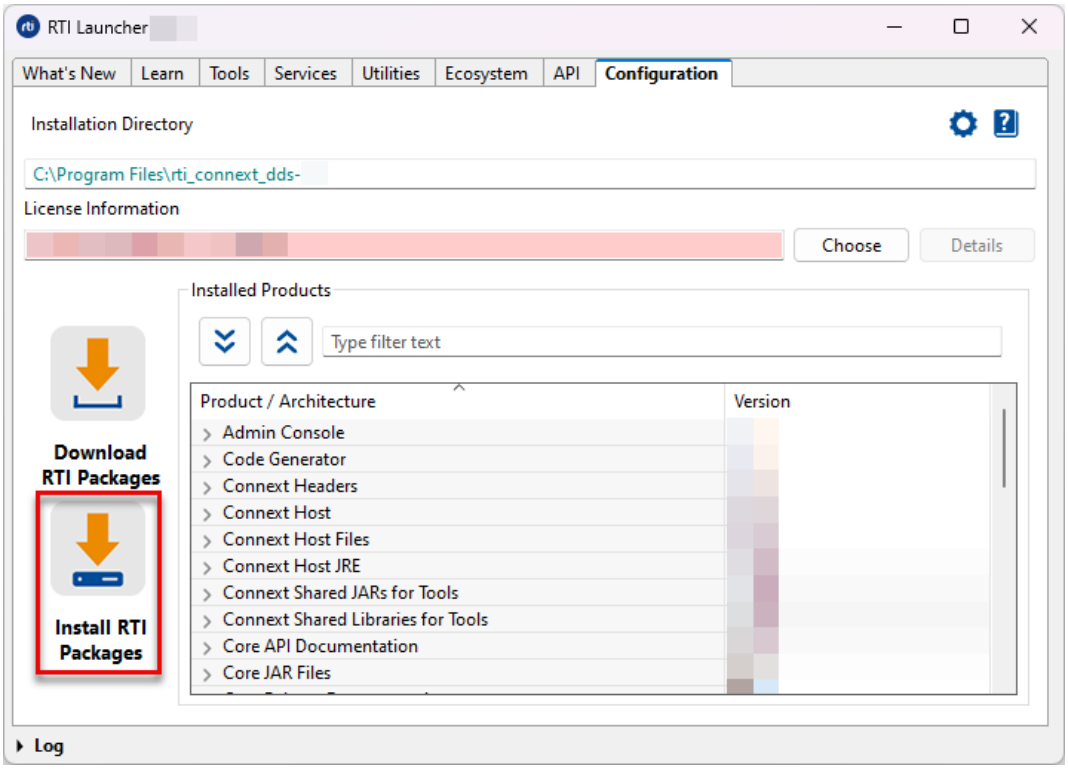
You can install *Connext Micro* from either the *RTI Launcher* GUI or a command line.

**Note:** You must first install *Connext Professional* or *Connext Drive* before installing *Connext Micro* packages. Check the *Compatibility* section for the right versions of these products to use, then refer to the RTI Connext Installation Guide or Installing Connext Drive for installation instructions.

### Installing with a GUI

After you install *Connext Professional*, you'll see an `rti_connext_dds-<version>` directory. In that directory, you'll have a tool called *RTI Launcher*.

To install the *Connext Micro* packages from the *Launcher* tool, open the Configuration tab, and select "Install RTI Packages." This will open a dialog that allows you to select one or more **.rtipkg** files that you would like to install.




Select the host and target packages in the dialog, then click **Install**.

Once installed, you will find a directory called `rti_connext_dds_micro-<version>` in the `rti_connext_dds-<version>` directory.

### Installing from a command line

To install *Connext Micro* from the command line, type:

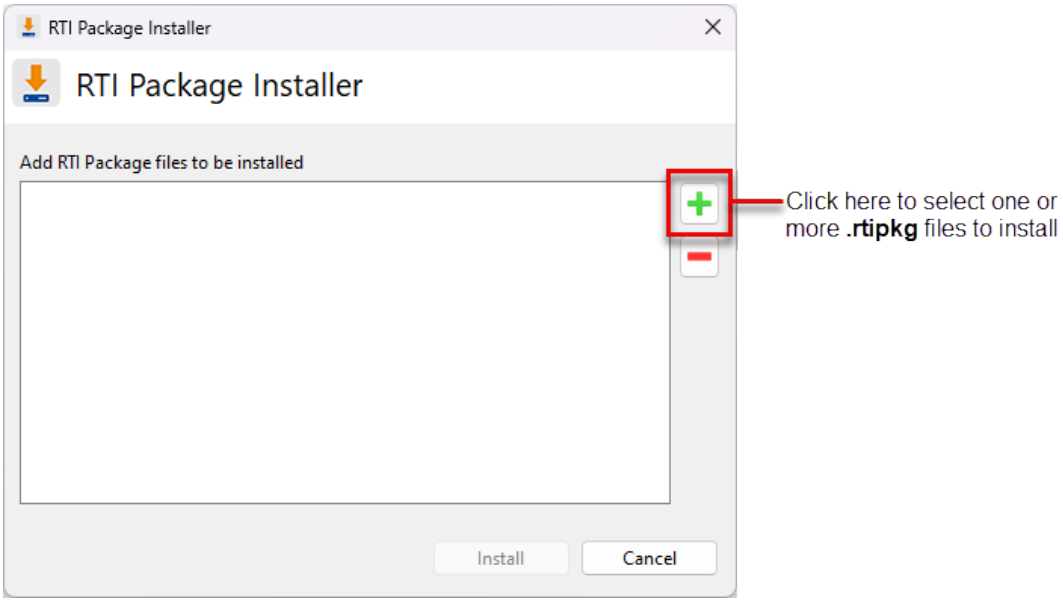
Linux

```
$ rti_connext_dds-<version>/bin/rtipkginstall <path to rtipkg>
```

macOS

```
$ rti_connext_dds-<version>/bin/rtipkginstall <path to rtipkg>
```

Windows




```
> rti_connext_dds-<version>\bin\rtipkginstall.bat <path to rtipkg>
```

Run this command for both the host and target `.rtipkg` files.

### Paths mentioned in documentation

This documentation refers to the following directories, depending on your operating system:

Linux

- `$NDDSHOME` This refers to the installation directory for *Connext*.
- `$RTIMEHOME` This refers to the installation directory for *Connext Micro*.

The default installation paths are:

- Non-root user:
  - *Connext*: `/home/<your user name>/rti_connext_dds-<version>`
  - *Connext Micro*: `/home/<your user name>/rti_connext_dds-<version>/rti_connext_dds_micro-<version>`
- Root user:
  - *Connext*: `/opt/rti_connext_dds-<version>`
  - *Connext Micro*: `/opt/rti_connext_dds-<version>/rti_connext_dds_micro-<version>`

`$NDDSHOME` is an environment variable set to the installation path for *Connext*. `$RTIMEHOME` is an environment variable set to the installation path for *Connext Micro*.

macOS

- `$NDDSHOME` This refers to the installation directory for *Connext*.
- `$RTIMEHOME` This refers to the installation directory for *Connext Micro*.

The default installation paths are:

- *Connext*: `/Applications/rti_connext_dds-<version>`
- *Connext Micro*: `/Applications/rti_connext_dds-<version>/rti_connext_dds_micro-<version>`

`$NDDSHOME` is an environment variable set to the installation path for *Connext*. `$RTIMEHOME` is an environment variable set to the installation path for *Connext Micro*.

Windows

- `%NDDSHOME%` This refers to the installation directory for *Connext*.
- `%RTIMEHOME%` This refers to the installation directory for *Connext Micro*.

The default installation paths are:

- User without Administrator privileges:
  - *Connext*: `<your home directory>\rti_connext_dds-<version>`
  - *Connext Micro*: `<your home directory>\rti_connext_dds-<version>\rti_connext_dds_micro-<version>`
- User with Administrator privileges:
  - *Connext*: `"C:\Program Files\rti_connext_dds-<version>"`
  - *Connext Micro*: `"C:\Program Files\rti_connext_dds-<version>\rti_connext_dds_micro-<version>"`

`%NDDSHOME%` is an environment variable set to the installation path for *Connext*. `%RTIMEHOME%` is an environment variable set to the installation path for *Connext Micro*.

---

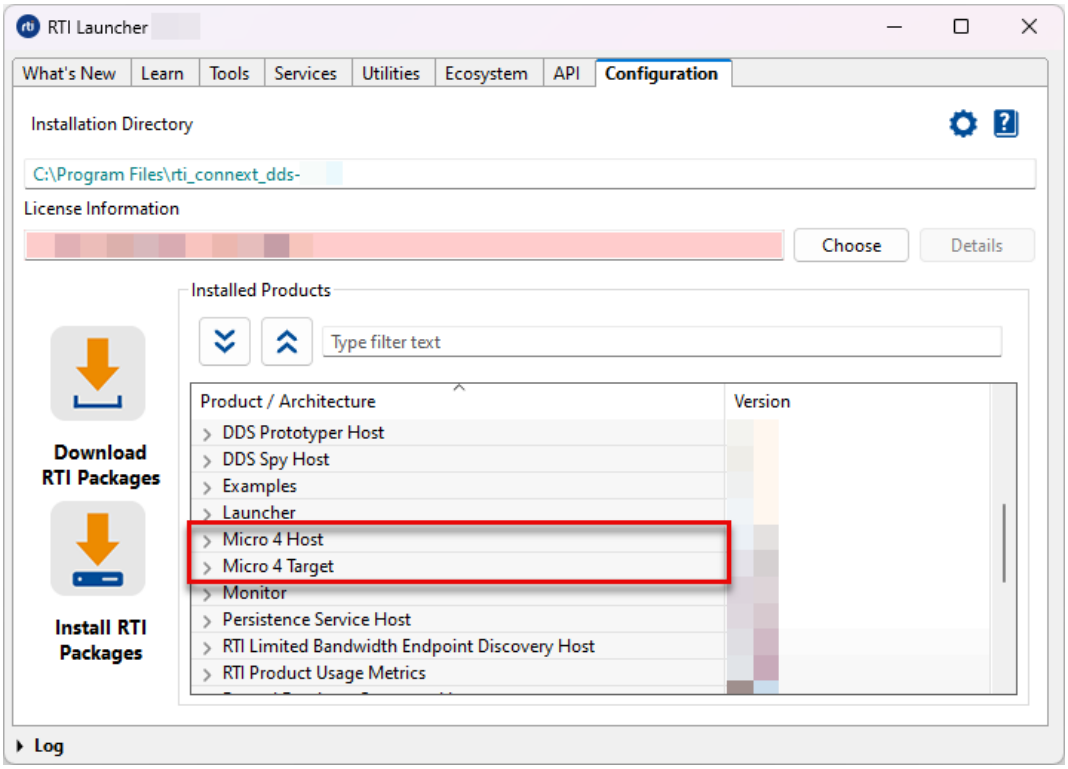
**Note:** When using a command prompt to enter a command that includes the path `C:\Program Files` (or any directory name that has a space), enclose the path in quotation marks. For example: `"C:\Program Files\rti_connext_dds-version\bin\rtilauncher.bat"`. Or if you have defined the `NDDSHOME` environment variable: `"%NDDSHOME%\bin\rtilauncher.bat"`.

---

Sometimes this documentation uses `<NDDSHOME>` to refer to the installation path for *Connext*, or `<RTIMEHOME>` for *Connext Micro*. Whenever you see `<NDDSHOME>` or `<RTIMEHOME>` used in a path, replace it with `$NDDSHOME` or `$RTIMEHOME` for Linux or macOS systems, with `%NDDSHOME%` or `%RTIMEHOME%` for Windows systems, or with your installation path.

### 1.1.4 Checking what is installed

To confirm that you have installed *Connext Micro*, you can use the *Launcher* tool. Once in *Launcher*, open the Configuration tab:




If you do not see both the *Connext Micro* host and target listed, you are missing one of the *Connext Micro* packages in your installation.

### 1.1.5 Where do I get more help?

Additional documentation and user forums can be found on community.rti.com.

Continue to *Hello World with Publish/Subscribe* to start learning about the capabilities and features of *Connext Micro*.

## 1.2 Hello World with Publish/Subscribe

In this section we are going to learn about the Publish/Subscribe model using a simple "Hello World" application. You will learn about the RTI *Code Generator* and how to create a *DataWriter* and a *DataReader* to publish and subscribe to data.

Publish/Subscribe is a communications model where data producers "publish" data and data consumers "subscribe" to data. These publishers and subscribers don't need to know about each other ahead of time; they discover each other dynamically at runtime. The data they share is described by a "topic," and publishers and subscribers send and receive data only for the topics they are

interested in. In this pattern, many publishers may publish the same topic, and many subscribers may subscribe to the same topic. Subscribers receive data from all of the publishers that they share a topic with. Publishers send data directly to subscribers, with no need for a broker or centralized application to mediate communications.

### 1.2.1 Introduction to DataWriters, DataReaders, and Topics

*Connext Micro* conforms to the OMG Data Distribution Service (DDS) connectivity standard. In DDS, the objects that actually publish data are called *DataWriters*, and the objects that subscribe to data are *DataReaders*. *DataWriters* and *DataReaders* are associated with a single *Topic* object that describes that data. (DDS also has *Publisher* and *Subscriber* objects, but we will talk about them later.) An application typically has a combination of *DataWriters* and *DataReaders*.

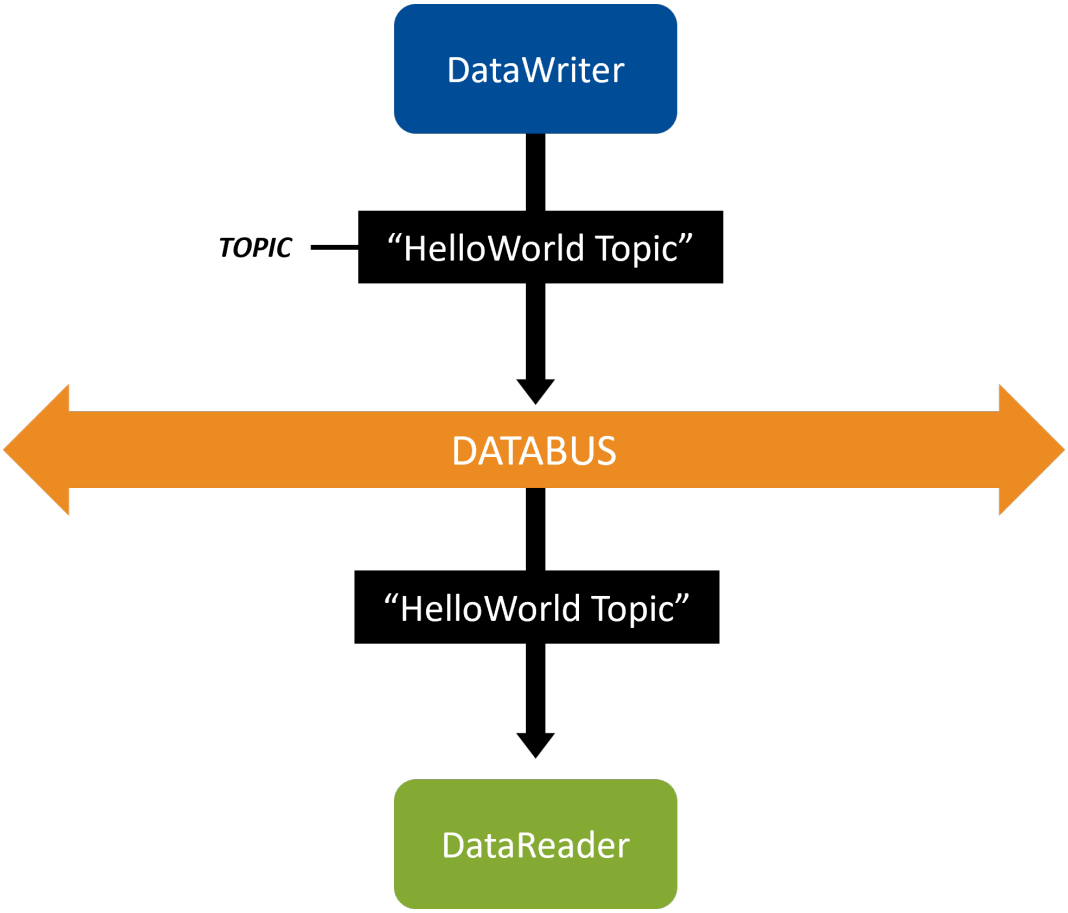



Figure 1.1: *DataWriters* write data and *DataReaders* read data of a *Topic*.

*Connext Micro* is responsible for discovering *DataWriters* and *DataReaders* in a system, checking if they have a matching *Topic* (and compatible quality of service, which we will discuss later) and then providing the communication between those *DataWriters* and *DataReaders*. Logically, this means you can visualize your applications as having *DataWriters* and *DataReaders* that connect to a "databus," because your applications are not specifying exactly which other applications they communicate with—they only specify which *Topics* they read from and write to the databus, and

*Connext Micro* sets up the communication. Note that there is no "databus" object or *Connext Micro* service deployed in your system—this is simply a logical way to visualize systems in which you don't have to configure each communication path.

### 1.2.2 Create a DataWriter and DataReader

We are going to start with a simple "Hello World" application in C to show how to use the *Code Generator*, and how to create a *DataWriter* and a *DataReader*.

---

**Tip:** By the end of this exercise, a publishing application will send data, and a subscribing application will receive and print it to the console.

---

#### Set environment variables

Before we begin, we need to set some environment variables:

1. Set the `RTIMEHOME` environment variable to the installation directory path for *RTI Connext Micro*.

   If you installed *Connext Professional* with default settings, *RTI Connext Micro* will be here: `<path_to_connext_dds_installation>/rti_connext_dds-<version>/rti_connext_micro-<version>`. If you copied *RTI Connext Micro* to another place, set `RTIMEHOME` to point to that location.

2. Add `<RTIMEHOME>/rtiddsgen/scripts` to your path environment variable folder.

3. Add `<RTIMEHOME>/resource/scripts` to your path environment variable folder.

#### Generate an example application

We'll use the RTI *Code Generator* (also referred to as *rtiddsgen*) included with *Connext Micro* to generate a DDS example application. To do that, we'll start with a type definition file as input.

#### Create a type definition file

Create a **HelloWorld.idl** file with the following type definition:

```
// Hello world!
struct HelloWorld {

    // String with maximum length of 256 characters
    string<256> msg;

};
```

In this file, the data `HelloWorld` is described in the language-independent Interface Definition Language (IDL). IDL allows you to declare data types used for communication. The RTI *Code Generator* (*rtiddsgen*) translates from this language-independent data type into code specific for your programming language. The generated code serializes and deserializes your data into and out of a network format.

### Run Code Generator

---

**Tip:** Before running *rtiddsgen*, make sure you added `rti_connext_dds_micro-<version>/rtiddsgen/scripts` to your path environment variable folder as described in *Set environment variables*. Run the following command to check:

```
rtiddsgen -version
```

You should see the following output:

```
INFO com.rti.ndds.nddsgen.Main rtiddsgen version <version>
INFO com.rti.ndds.nddsgen.Main Connext DDS templates: XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-
↪XXXX
INFO com.rti.ndds.nddsgen.Main Connext Micro templates: XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-
↪XXXX-XXXX
INFO com.rti.ndds.nddsgen.Main Done
```

If you don't see `Connext Micro templates:` in the output, you might be using a version of *rtiddsgen* from *Connext Professional*. Make sure you added *rtiddsgen* to the environment path from `rti_connext_dds_micro-<version>/rtiddsgen/scripts`.

---

From a terminal or command prompt, change directories to where you created **HelloWorld.idl** and run the following command:

Linux

```
$ rtiddsgen -example -language C HelloWorld.idl
```

macOS

```
$ rtiddsgen -example -language C HelloWorld.idl
```

Windows

```
> rtiddsgen -example -language C -ppDisable HelloWorld.idl
```

`-ppDisable` disables the preprocessor. It is necessary for running *rtiddsgen* on a Windows system if the preprocessor is not in your path. You can only use `-ppDisable` if your IDL does not contain any C preprocessor directives, as here—otherwise you must add the preprocessor to your path.

### Overview of generated and example code

The code you just generated includes the files in Table 1.1.

Table 1.1: Generated and Example Code Files

| Files | Description |
|---|---|
| • HelloWorld.h<br>• HelloWorld.c<br>• HelloWorldPlugin.h<br>• HelloWorldPlugin.c<br>• HelloWorldSupport.h<br>• HelloWorldSupport.c | The C definition of your data type, and the code used to serialize and deserialize it (convert it to a format for the network). This is the type-specific code that will be used in your real application. These files were generated when you ran *rtiddsgen*. |
| • HelloWorldApplication.h<br>• HelloWorldApplication.c<br>• HelloWorld_publisher.c<br>• HelloWorld_subscriber.c | Example application code. It includes code for two applications you can read and modify. These will compile into separate applications, **HelloWorld_publisher** and **HelloWorld_subscriber**. These files were generated when you ran *rtiddsgen*. |
| • CMakeLists.txt | Used to compile your application with CMake. This file was generated when you ran *rtiddsgen*. |
| • README.txt | Instructions for how to open and modify the files, compile, and run the example. This file was generated when you ran *rtiddsgen*. |

### Modify application

1. In your example application directory, open `HelloWorldApplication.c` in your IDE of choice.

   This snippet shows how to create a *Topic* (with a name and data type):

```
sprintf(application->topic_name, "Example HelloWorld");
application->topic = DDS_DomainParticipant_create_topic(
    application->participant,
    application->topic_name,
    application->type_name,
    &DDS_TOPIC_QOS_DEFAULT,
    NULL,
    DDS_STATUS_MASK_NONE);
if (application->topic == NULL)
{
    printf("topic == NULL\n");
    goto done;
}
```

2. Change the *Topic* name from "Example HelloWorld" to "HelloWorld Topic":

```
sprintf(application->topic_name, "HelloWorld Topic");
application->topic = DDS_DomainParticipant_create_topic(
    application->participant,
    application->topic_name,
    application->type_name,
    &DDS_TOPIC_QOS_DEFAULT,
    NULL,
    DDS_STATUS_MASK_NONE);
if (application->topic == NULL)
{
    printf("topic == NULL\n");
    goto done;
}
```

### Modify publisher

1. Open `HelloWorld_publisher.c` in your IDE of choice.

   The following snippet shows how to write a HelloWorld update using the *DataWriter*'s write method:

```
for (i = 0; (application->count <= 0) || (i < application->count); ++i)
{

    /* TODO set sample attributes here */

    retcode = HelloWorldDataWriter_write(
        hw_datawriter,
        sample,
```

(continues on next page)

(continued from previous page)

```
        &DDS_HANDLE_NIL);
    if (retcode != DDS_RETCODE_OK)
    {
        printf("Failed to write sample\n");
    }
    else
    {
        printf("Written sample %d\n",(i+1));
    }

    OSAPI_Thread_sleep((RTI_UINT32)application->sleep_time);
}
```

2. Add the following code to send the message "Hello world!" with a count:

```
for (i = 0; (application->count <= 0) || (i < application->count); ++i)
{

    /* TODO set sample attributes here */

    snprintf(sample->msg,128,"Hello World (%d) ! \n",i);

    retcode = HelloWorldDataWriter_write(
        hw_datawriter,
        sample,
        &DDS_HANDLE_NIL);
    if (retcode != DDS_RETCODE_OK)
    {
        printf("Failed to write sample\n");
    }
    else
    {
        printf("Written sample %d\n",(i+1));
    }

    OSAPI_Thread_sleep((RTI_UINT32)application->sleep_time);
}
```

Recall that your "HelloWorld Topic" describes your data. This *Topic* is associated with the data type **HelloWorld**, which is defined in the IDL file (see *Run Code Generator*). The data type **HelloWorld** contains a string field named **msg**. In this step, you have just added code to set a value for the msg field. Now, when the *DataWriter* writes data, the msg field in the data will contain the string "Hello world! 1", "Hello world! 2", etc.

### Definition

A **sample** is a single update to a *Topic*, such as "Hello world (1) !". Every time an application calls `write()`, it is "writing a sample." Every time an application receives data, it is "receiving a sample."

Note that samples don't necessarily overwrite each other. For example, if you set up a

RELIABLE Quality of Service (QoS)[1] with a History **kind** of KEEP_ALL, all samples will be saved and accumulate.

### Modify subscriber

1. Open `HelloWorld_subscriber.c` in your IDE of choice.

   The following snippet shows how to process a sample:

```
for (i = 0; i < HelloWorldSeq_get_length(&sample_seq); ++i)
   {
      sample_info = DDS_SampleInfoSeq_get_reference(&info_seq, i);

      if (sample_info->valid_data)
      {
            sample = HelloWorldSeq_get_reference(&sample_seq, i);
            printf("\nValid sample received\n");
            *total_samples += 1;

            /* TODO read and process sample attributes here */
            (void)sample;

      }
      else
      {
            printf("\nSample received\n\tINVALID DATA\n");
      }
   }
```

2. Add the following code to display the received messages:

```
for (i = 0; i < HelloWorldSeq_get_length(&sample_seq); ++i)
   {
      sample_info = DDS_SampleInfoSeq_get_reference(&info_seq, i);

      if (sample_info->valid_data)
      {
            sample = HelloWorldSeq_get_reference(&sample_seq, i);
            printf("\nValid sample received\n");
            *total_samples += 1;

            /* TODO read and process sample attributes here */
            printf("%s\n", sample->msg);

      }
      else
      {
            printf("\nSample received\n\tINVALID DATA\n");
      }
   }
```

[1] For more information on QoS policies, refer to QoS Policies in the API Reference.

### Compile your changes

Now that you have made changes to the example code, compile the code with your modifications. Run the following command from the example directory:

Linux

```
$ rtime-make --config Debug --build --target x86_64leElfgcc13.3.0-Linux6 --source-dir . -
↪G "Unix Makefiles" --delete
```

macOS

```
$ rtime-make --config Debug --build --target armv8leMachOclang15.0-Darwin23 --source-dir␣
↪. -G "Unix Makefiles" --delete
```

Windows

```
> rtime-make.bat --config Debug -A x64 --target self --name x86_64lePEvs2017-Win10 --
↪build --source-dir .
```

After running the command, you should see two applications in the `objs/<architecture>` or `objs/<architecture>/Debug` directory:

- HelloWorld_publisher
- HelloWorld_subscriber

### Run the applications

---

**Note:** By default, this example uses two interfaces to receive samples. Since your installation might use different names for these interfaces (which would prevent communication), we'll use the option `-udp_intf <interface name>` to specify them in the following commands. You can use command `ifconfig` (on Linux or macOS systems) or `ipconfig` (on Windows systems) to know the name of all available interfaces.

---

1. From within the `objs/<architecture>` or `objs/<architecture>/Debug` directory, enter the following full path:

   Linux

   ```
   $ objs/x86_64leElfgcc13.3.0-Linux6/Debug/HelloWorld_publisher -udp_intf <interface␣
   ↪name>
   ```

   macOS

   ```
   $ objs/armv8leMachOclang15.0-Darwin23/Debug/HelloWorld_publisher -udp_intf
   ↪<interface name>
   ```

   Windows

```
> objs\x86_64lePEvs2017-Win10\Debug\HelloWorld_publisher.exe -udp_intf <interface␣
↪name>
```

You should see this in the window for the publisher:

```
Written sample 1
Written sample 2
Written sample 3
Written sample 4

...
```

2. Open another command prompt window, and from within the example directory, enter the following full path:

   Linux

   ```
   $ objs/<architecture>/Debug/HelloWorld_subscriber -udp_intf <interface name>
   ```

   macOS

   ```
   $ objs/<architecture>/Debug/HelloWorld_subscriber -udp_intf <interface name>
   ```

   Windows

   ```
   > objs\<architecture>\Debug\HelloWorld_subscriber.exe -udp_intf <interface name>
   ```

   You should see this in the window for the subscriber:

   ```
   Valid sample received
   Hello World (1) !

   Subscriber sleeping for 1000 msec...

   Valid sample received
   Hello World (2) !

   Subscriber sleeping for 1000 msec...

   Valid sample received
   Hello World (3) !

   Subscriber sleeping for 1000 msec...

   Valid sample received
   Hello World (4) !

   ...
   ```

   The `Hello world <count> !` line is the data being sent by the *DataWriter*. If the *DataWriter* weren't communicating with the *DataReader*, you would just see the `Subscriber sleeping for 1000 msec...` lines and not the "msg" lines. (The subscribing application prints the

"timed out" lines after the WaitSet times out while waiting for data, then it prints the "msg:" lines when it receives data from the *DataWriter*.)

### Taking it further

Under the hood, the publishing and subscribing applications are doing a lot of work:

Before communication starts, the *DataWriter* and *DataReader* discover each other and check that they have the same *Topic* name, compatible data types, and compatible QoS. After discovery, the *DataWriter* sends data directly to the *DataReader*, with no message broker required.

When you run the applications on the same machine, by default they communicate over shared memory. If you run one on another machine, they communicate over the network using UDP.

### Start up multiple publishing or subscribing applications

Try starting up multiple publishing or subscribing applications, and you will see that they will also send or receive data. (Remember to run from the example directory.)

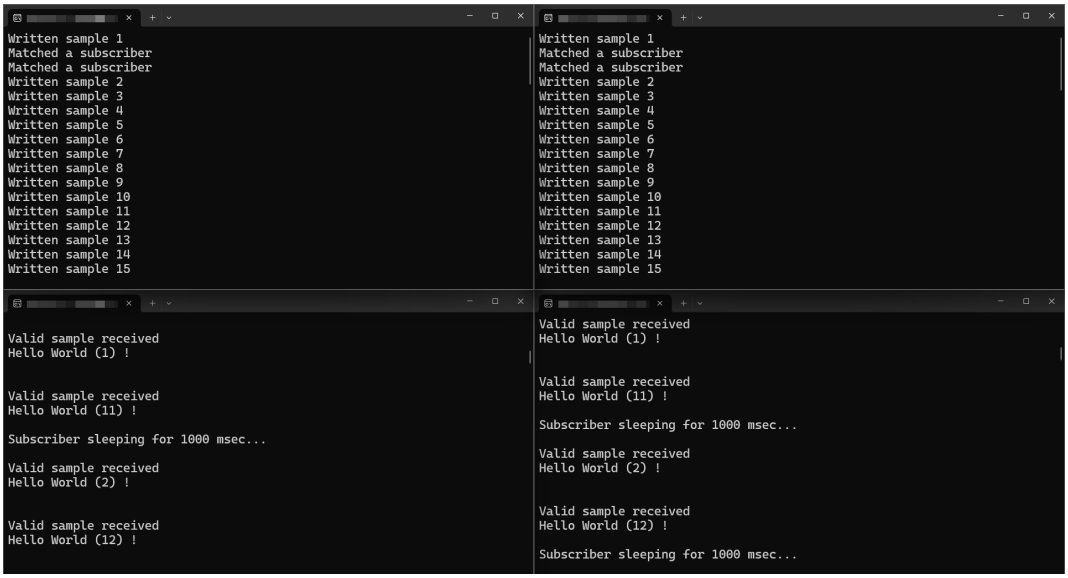



Figure 1.2: Two applications publishing, and two subscribing, to the same *Topic*. Notice that each subscriber is receiving data from both publishers.

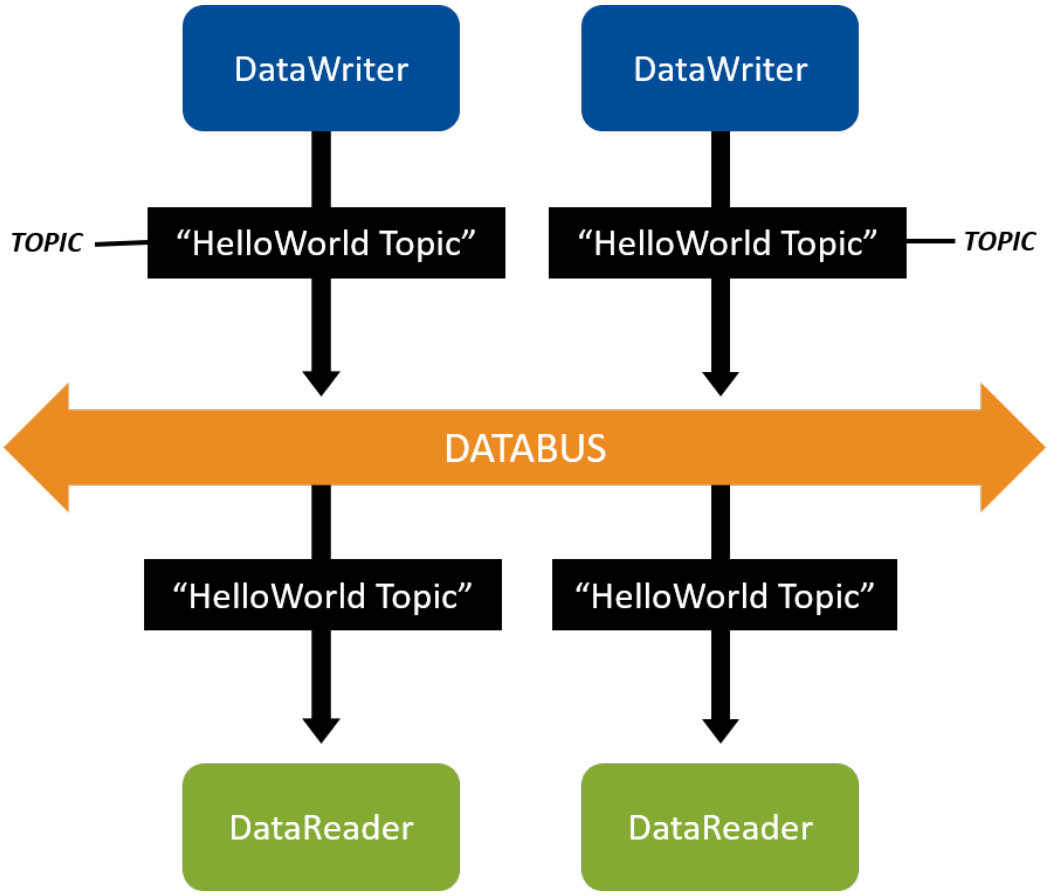



Figure 1.3: Two applications publishing, and two subscribing, to the same *Topic*.

### 1.2.3 Troubleshooting

#### Why aren't my applications communicating?

If you are running both applications and they aren't communicating (you don't see the `Hello world <count> !` lines shown at the end of *Run the applications*), here are some things to check:

- Make sure you specified the correct interfaces for both your applications with the `-udp_intf` option, as described in *Run the applications*.
- Check if your interface supports local loopback. If it doesn't, try using an interface that does, or run the application between two host machines and change the peer on each one to be the host address of the other machine.
- See if your machine uses a firewall. You may need to disable your firewall or configure it to allow multicast traffic for communication to be established.

#### Why does the DataReader miss the first samples?

Discovery is not an instantaneous event. It takes some time for the discovery process between applications to complete. The *DataWriter* and *DataReader* must discover each other before they can start communicating. Therefore, if you send data immediately after creating the *Connext Micro* entities, *DataReaders* will not receive the first few samples because they are still in-process of discovering the *DataWriters* and vice versa. This is true even when the *DataWriters* and *DataReaders* are reliable, because the Reliability QoS Policy on its own does not guarantee delivery to *DataReaders* that have not been discovered yet.

You can overcome this behavior with the Durability QoS Policy, which can be set to deliver historical samples (that *DataWriters* already sent) to late-joining *DataReaders*. The *DataReaders* will then receive the first samples they originally missed.

### 1.2.4 Next steps

Congratulations! You've written your first *Connext Micro* application. You've experienced a quick overview of the development process from defining a data type and using the RTI *Code Generator*, to building an example application.

Continue to *Developing Applications* for a deeper look at the process of developing your own *Connext Micro* applications.

# Chapter 2

# Developing Applications

This section describes how to write *Connext Micro* applications. It covers preparing your development environment, defining data types, generating support code for your data types, and creating the entities that publish and subscribe to data.

For a deeper dive into *Connext Micro*'s features, refer to the *Programmer's Guide*.

## 2.1 Prepare Your Development Environment

This section describes how to set up your development environment for *Connext Micro* applications, such as the required environment variables, compilers, compiler definitions, and libraries.

### 2.1.1 Set environment variables

The `RTIMEHOME` environment variable must be set to the installation directory path for *RTI Connext Micro*. If you installed *RTI Connext* with default settings, *RTI Connext Micro* will be here: `<path_to_connext_dds_installation>/rti_connext_dds-<version>/rti_connext_micro-<version>`. If you copied *RTI Connext Micro* to another place, set `RTIMEHOME` to point to that location.

### 2.1.2 Add required preprocessor flags

All *Connext Micro* applications require the following preprocessor defines:

```
-IRTIMEHOME/include
-IRTIMEHOME/include/rti_me
```

Add the following preprocessor defines, according to your platform and compiler:

Windows

Using MSVSCC:

```
-DOSAPI_CC_DEF_H=osapi/osapi_cc_msvsc.h
-DRTI_WIN32
```

macOS

Using clang:

```
-DOSAPI_CC_DEF_H=osapi/osapi_cc_clang.h
-DRTI_DARWIN
```

Linux

Using GCC:

```
-DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h
-DRTI_LINUX
```

QNX

Using QCC:

```
-DOSAPI_CC_DEF_H=osapi/osapi_cc_qcc.h
-DRTI_QNX
```

### 2.1.3 Link applications and libraries

Add the library path for both the PIL and PSL to the linker's search path:

- `RTIMEHOME/lib/<arch>/` (PIL)
- `RTIMEHOME/lib/<arch>-<PSL>/` (PSL)

---

**Note:** When executing executables that are linked with the *Connext Micro* shared libraries, you must add the path to the PIL architecture directory to the runtime linker's search path.

---

To link a C application, the libraries are required in the following order:

- `RTIMEHOME/lib/<arch>/`
    1. `rti_me_appgen` (if using the *Application Generation Using XML* plugin)
    2. `rti_me_netiosdm` (if using *Zero Copy Transfer* with the *Shared Memory Transport (SHMEM)*)
    3. `rti_me_netiozcopy` (if using the *Zero Copy v2 Transport*)
    4. `rti_me_netioshmem` (if using the *Shared Memory Transport (SHMEM)*)
    5. `rti_me_discdpde` (if using DPDE)
    6. `rti_me_discdpse` (if using DPSE)
    7. `rti_me_ddsfilter` (if using *Content Filtering*)

8. `rti_me_ddsxtypes` (if using XTypes)
9. `rti_me_rhsm`, `rti_me_whsm`, and `rti_me` (always required)

- `RTIMEHOME/lib/<arch>-<PSL>/`

10. `rti_me_netiopsl` (when building with a Platform Independent Library)
11. `rti_me_ospsl` (when building with a Platform Independent Library)

To link a C++ application, the libraries are required in the following order:

- `RTIMEHOME/lib/<arch>/`

1. `rti_me_appgen` (if using the *Application Generation Using XML* plugin)
2. `rti_me_ddsxtypes` (if using XTypes)
3. `rti_me_cpp`, `rti_me_netiosdm`, `rti_me_netiozcopy`, `rti_me_netioshmem`, `rti_me_discdpde`, `rti_me_discdpse`, `rti_me_ddsfilter`[1], `rti_me_rhsm`, `rti_me_whsm`, and `rti_me` (always required)

- `RTIMEHOME/lib/<arch>-<PSL>/`

4. `rti_me_netiopsl_cpp` (when building with a Platform Independent Library)
5. `rti_me_netiopsl` (when building with a Platform Independent Library)
6. `rti_me_ospsl` (when building with a Platform Independent Library)

## 2.2 Define a Data Type

To distribute data using *Connext Micro*, you must first define a data type, then run the *rtiddsgen* IDL compiler. *rtiddsgen* will generate the type-specific support code that *Connext Micro* needs to publish and subscribe to that data type.

*Connext Micro* accepts types definitions in Interface Definition Language (IDL) format.

For instance, the HelloWorld examples provided with *Connext Micro* use this simple type, which contains a string "msg" with a maximum length of 128 characters, excluding the `NUL` terminator.

```
struct HelloWorld
{
    string<128> msg;
};
```

For more details, see *Data Types*.

[1] Only required if using *Content Filtering*.

## 2.3 Generate Type Support Code with rtiddsgen

You will provide your IDL as an input to *rtiddsgen*. *rtiddsgen* supports code generation for the following standard types:

- octet, char, wchar
- short, unsigned short
- long, unsigned long
- long long, unsigned long long float
- double, long double
- boolean
- string
- struct
- array
- enum
- wstring
- sequence
- union
- typedef
- value type

*rtiddsgen* is in *<your_top_level_dir>/rti_connext_dds-<version>/rti_connext_micro-<version>/bin*.

To generate support code for data types in a file called HelloWorld.idl:

```
rtiddsgen -micro -language C -replace HelloWorld.idl
```

Run `rtiddsgen -help` to see all available options. For the options used here:

- The `-micro` option is necessary to generate support code specific to *Connext Micro*, as *rtiddsgen* is also capable of generating support code for *Connext*, and the generated code for the two are different. Note that *Connext Micro* and *RTI Connext Cert* use the same *rtiddsgen* and similar code is generated. However, when the generated code is compiled with RTI_CERT certain APIs are excluded.
- The `-language` option specifies the language of the generated code. *Connext Micro* supports C and traditional C++ (with `-language C++`).
- The `-replace` option specifies that the new generated code will replace, or overwrite, any existing files with the same name.

*rtiddsgen* generates the following files for an input file HelloWorld.idl:

- **HelloWorld.h and HelloWorld.c**. Operations to manage a sample of the type, and a DDS sequence of the type.
- **HelloWorldPlugin.h and HelloWorldPlugin.c**. Implements the type-plugin interface defined by *Connext Micro*. Includes operations to serialize and deserialize a sample of the type and its DDS instance keys.
- **HelloWorldSupport.h and HelloWorldSupport.c**. Operations to generate a type-specific *DataWriter* and *DataReader*, and to register the type with a DDS *DomainParticipant*.

***Connext Micro* supports generating two different types of type support:**

1. XTypes - This is the default method and generates type support for types compatible with the DDS XTypes standard.
2. CDR - This is the original CDR support and uses less memory. To enable this method, pass the option `-intepreted 0` to *rtiddsgen*.

For more details, see *Generating Type Support with rtiddsgen*.

## 2.4 Create an Application

The rest of this guide will walk you through the steps to create an application and will provide example code snippets. It assumes that you have defined your types (see *Define a Data Type*) and have used *rtiddsgen* to generate their support code (see *Generate Type Support Code with rtiddsgen*).

### 2.4.1 Registry Configuration

The *Connext Micro* architecture defines a run-time (RT) component interface that provides a generic framework for organizing and extending functionality of an application. An RT component is created and deleted with an RT component factory. Each RT component factory must be registered within an RT registry in order for its components to be usable by an application.

The DomainParticipantFactory, in addition to its standard role of creating and deleting *DomainParticipants*, contains the RT registry where an application registers the components it will use.

Every DDS application must register these components:

- **Writer History**. Queue of written samples of a *DataWriter*. Must be registered with the name `DDSHST_WRITER_DEFAULT_HISTORY_NAME`.
- **Reader History**. Queue of received samples of a *DataReader*. Must be registered with the name `DDSHST_READER_DEFAULT_HISTORY_NAME`.

In addition, an application typically registers a discovery plugin in order to discover other DDS applications on a network:

- **Discovery (DPDE or DPSE)**. Discovery component. Choose either dynamic (DPDE) or static (DPSE) endpoint discovery.

Example source:

- Get the RT Registry from the DomainParticipantFactory singleton:

```
DDS_DomainParticipantFactory *factory = NULL;
RT_Registry_T *registry = NULL;

factory = DDS_DomainParticipantFactory_get_instance();
registry = DDS_DomainParticipantFactory_get_registry(factory);
```

- Register the Writer History and Reader History components with the registry:

```
/* Register Writer History */
if (!RT_Registry_register(registry, DDSHST_WRITER_DEFAULT_HISTORY_NAME,
                          WHSM_HistoryFactory_get_interface(), NULL, NULL))
{
    /* failure */
}

/* Register Reader History */
if (!RT_Registry_register(registry, DDSHST_READER_DEFAULT_HISTORY_NAME,
                          RHSM_HistoryFactory_get_interface(), NULL, NULL))
{
    /* failure */
}
```

Only one discovery component can be registered, either DPDE or DPSE. Each has its own properties that can be configured upon registration.

- Register DPDE for dynamic participant, dynamic endpoint discovery:

```
struct DPDE_DiscoveryPluginProperty discovery_plugin_properties =
    DPDE_DiscoveryPluginProperty_INITIALIZER;

/* Configure properties */
discovery_plugin_properties.participant_liveliness_assert_period.sec = 5;
discovery_plugin_properties.participant_liveliness_assert_period.nanosec = 0;
discovery_plugin_properties.participant_liveliness_lease_duration.sec = 30;
discovery_plugin_properties.participant_liveliness_lease_duration.nanosec = 0;

/* Register DPDE with updated properties. The name is arbitrary, but
 * cannot exceed 7 ASCIIZ characters.
if (!RT_Registry_register(registry,
                          "dpde",
                          DPDE_DiscoveryFactory_get_interface(),
                          &discovery_plugin_properties._parent,
                          NULL))
{
    /* failure */
}
```

- Register DPSE for dynamic participant, static endpoint discovery:

```
struct DPSE_DiscoveryPluginProperty discovery_plugin_properties =
    DPSE_DiscoveryPluginProperty_INITIALIZER;

/*  Configure properties */
discovery_plugin_properties.participant_liveliness_assert_period.sec = 5;
discovery_plugin_properties.participant_liveliness_assert_period.nanosec = 0;
discovery_plugin_properties.participant_liveliness_lease_duration.sec = 30;
discovery_plugin_properties.participant_liveliness_lease_duration.nanosec = 0;

/* Register DPSE with updated properties. The name is arbitrary, but
 * cannot exceed 7 ASCIIZ characters.
 */
if (!RT_Registry_register(registry,
                          "dpse",
                          DPSE_DiscoveryFactory_get_interface(),
                          &discovery_plugin_properties._parent,
                          NULL))
{
    printf("failed to register dpse\n");
    goto done;
}
```

For more information, see *Application Generation Using XML*.

## 2.5 Configure UDP Transport

*RTI Connext Micro* comes with several transports. In this example we'll use the UDP transport and register it with the *Connext Micro* run-time.

- Configure the UDP transport properties:

```
struct UDP_InterfaceFactoryProperty *udp_property = NULL;

udp_property = (struct UDP_InterfaceFactoryProperty *)
    malloc(sizeof(struct UDP_InterfaceFactoryProperty));

if (udp_property != NULL)
{
    *udp_property = UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;

    /* allow_interface: Names of network interfaces allowed to receive.
     * Allow one loopback (lo) and one NIC (eth0).
     */
    REDA_StringSeq_set_maximum(&udp_property->allow_interface,2);
    REDA_StringSeq_set_length(&udp_property->allow_interface,2);

    *REDA_StringSeq_get_reference(&udp_property->allow_interface,0) = DDS_String_
↪dup("lo");
    *REDA_StringSeq_get_reference(&udp_property->allow_interface,1) = DDS_String_
↪dup("eth0");
```

(continues on next page)

(continued from previous page)

```
}
else
{
    /* failure */
}
```

- Register the UDP transport with updated properties:

```
if (!UDP_Interface_register(registry, "_udp",udp_property))
{
    /* failure */
}
```

For more details, see *Datagram UDP Setup*.

## 2.6 Create DomainParticipant, Topic, and Type

A DomainParticipantFactory creates *DomainParticipants*, and a *DomainParticipant* itself is the factory for creating *Publishers*, *Subscribers*, and *Topics*.

When creating a *DomainParticipant*, you may need to customize DomainParticipantQos, notably for:

- **Resource limits**. Default resource limits are set at minimum values.
- **Initial peers**.
- **Discovery**. The name of the registered discovery component ("dpde" or "dpse") must be assigned to DiscoveryQosPolicy's name. Please note that in *Connext Cert*, only the DPSE discovery plugin is supported.
- **Participant Name**. Every *DomainParticipant* is given the same default name. Must be unique when using DPSE discovery.

Example code:

- Create a *DomainParticipant* with configured DomainParticipantQos:

```
DDS_DomainParticipant *participant = NULL;
struct DDS_DomainParticipantQos dp_qos =
                            DDS_DomainParticipantQos_INITIALIZER;

/* DDS domain of DomainParticipant */
DDS_Long domain_id = 0;

/* Name of your registered Discovery component. The name must match the
 * name the discovery plugin was registered with.
 */
if (!RT_ComponentFactoryId_set_name(&dp_qos.discovery.discovery.name,
                                    "dpde"))
{
```

(continues on next page)

(continued from previous page)

```
    /* failure */
}

/* Initial peers: use only default multicast peer
 */
DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers,1);
DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers,1);
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers,0) =
    DDS_String_dup("_udp://239.255.0.1");

/* Configure discovery to use the UDP transport and listen on all allowed
 * interfaces and the multicast address 239.255.0.1
 */
DDS_StringSeq_set_maximum(&dp_qos.discovery.enabled_transports,2);
DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports,2);
*DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports,0) =
        DDS_String_dup("_udp://");
*DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports,1) =
        DDS_String_dup("_udp://239.255.0.1");

/* Configure user-data to use UDP transport and receive on all allowed
 * interfaces
 */
DDS_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports,1);
*DDS_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports,0) =
        DDS_String_dup("_udp://");

/* Resource limits to discover up to 4 remote participants */
dp_qos.resource_limits.local_topic_allocation = 1;
dp_qos.resource_limits.local_type_allocation = 1;
dp_qos.resource_limits.local_reader_allocation = 1;
dp_qos.resource_limits.local_writer_allocation = 1;
dp_qos.resource_limits.remote_participant_allocation = 4;
dp_qos.resource_limits.remote_reader_allocation = 4;
dp_qos.resource_limits.remote_writer_allocation = 4;

/* Participant name */
strcpy(dp_qos.participant_name.name, "Participant_1");

participant =
    DDS_DomainParticipantFactory_create_participant(factory,
                                                    domain_id,
                                                    &dp_qos,
                                                    NULL,
                                                    DDS_STATUS_MASK_NONE);
 if (participant == NULL)
 {
    /* failure */
 }
```

### 2.6.1 Register Type

Your data types that have been generated from IDL need to be registered with the *DomainParticipants* that will be using them. Each registered type must have a unique name.

```
DDS_ReturnCode_t retcode;

retcode = DDS_DomainParticipant_register_type(participant,
                                              "HelloWorld",
                                              HelloWorldTypePlugin_get());
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}
```

### 2.6.2 Create Topic of Registered Type

DDS *Topics* encapsulate the types being communicated, and you can create *Topics* for your type once your type is registered.

A topic is given a name at creation (e.g. "Example HelloWorld"). The type associated with the *Topic* is specified with its registered name.

```
DDS_Topic *topic = NULL;

topic = DDS_DomainParticipant_create_topic(participant,
                                           "Example HelloWorld",
                                           "HelloWorld",
                                           &DDS_TOPIC_QOS_DEFAULT,
                                           NULL,
                                           DDS_STATUS_MASK_NONE);

if (topic == NULL)
{
    /* failure */
}
```

### 2.6.3 DPSE Discovery: Assert Remote Participant

DPSE Discovery relies on the application to specify the remote *DomainParticipants* that its local *DomainParticipants* are allowed to discover. Your application must call a DPSE API for each remote participant to be discovered. The API takes as input the name of the remote participant.

```
/* Allow discovery of remote participant with name Participant_2 */
retcode = DPSE_RemoteParticipant_assert(participant, "Participant_2");
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}
```

For more information, see *DDS Domains*.

## 2.7 Create Publisher

A DDS *Publisher* contains zero or more *DataWriters*. A publishing application must create at least one DDS *Publisher* and then a *DataWriter* for each *Topic* it wants to publish.

Please refer to PublisherQos and DataWriterQos for customizable policies for these DDS entities.

- Create *Publisher*:

```
DDS_Publisher *publisher = NULL;
publisher = DDS_DomainParticipant_create_publisher(participant,
                                                   &DDS_PUBLISHER_QOS_DEFAULT,
                                                   NULL,
                                                   DDS_STATUS_MASK_NONE);
if (publisher == NULL)
{
    /* failure */
}
```

For more information, see *Sending Data*.

## 2.8 Create DataWriter

```
DDS_DataWriter *datawriter = NULL;
struct DDS_DataWriterQos dw_qos = DDS_DataWriterQos_INITIALIZER;
struct DDS_DataWriterListener dw_listener = DDS_DataWriterListener_INITIALIZER;

/* Configure writer Qos */

/* Setting the rtps_object_id is only required when using DPSE */
dw_qos.protocol.rtps_object_id = 100;
dw_qos.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;
dw_qos.resource_limits.max_samples_per_instance = 2;
dw_qos.resource_limits.max_instances = 2;
dw_qos.resource_limits.max_samples =
    dw_qos.resource_limits.max_samples_per_instance * dw_qos.resource_limits.max_
↪instances;
dw_qos.history.depth = 1;
dw_qos.durability.kind = DDS_VOLATILE_DURABILITY_QOS;
dw_qos.protocol.rtps_reliable_writer.heartbeat_period.sec = 0;
dw_qos.protocol.rtps_reliable_writer.heartbeat_period.nanosec = 250000000;

/* Set enabled listener callbacks */
dw_listener.on_publication_matched = HelloWorldPublisher_on_publication_matched;

datawriter =
    DDS_Publisher_create_datawriter(publisher,
```

(continues on next page)

(continued from previous page)

```
                                    topic,
                                    &dw_qos,
                                    &dw_listener,
                                    DDS_PUBLICATION_MATCHED_STATUS);
if (datawriter == NULL)
{
    /* failure */
}
```

The DataWriterListener has its callbacks selectively enabled by the DDS status mask. In the example, the mask has set the on_publication_matched status, and accordingly the DataWriterListener has its on_publication_matched assigned to a callback function.

```
void HelloWorldPublisher_on_publication_matched(void *listener_data,
                                                DDS_DataWriter * writer,
                                                const struct DDS_
↪PublicationMatchedStatus *status)
{
    /* Print on match/unmatch */
    if (status->current_count_change > 0)
    {
        printf("Matched a subscriber\n");
    }
    else
    {
        printf("Unmatched a subscriber\n");
    }
}
```

### 2.8.1 DPSE Discovery: Assert Remote Subscription

A publishing application using DPSE discovery must specify the other *DataReaders* that its *DataWriters* are allowed to discover. Like the API for asserting a remote participant, the DPSE API for asserting a remote subscription must be called for each remote *DataReader* that a *DataWriter* may discover.

Whereas asserting a remote participant requires only the remote *Participant*'s name, asserting a remote subscription requires more configuration, as all QoS policies of the subscription necessary to determine matching must be known and thus specified.

```
struct DDS_SubscriptionBuiltinTopicData rem_subscription_data =
    DDS_SubscriptionBuiltinTopicData_INITIALIZER;

/* Set Reader's protocol.rtps_object_id */
rem_subscription_data.key.value[DDS_BUILTIN_TOPIC_KEY_OBJECT_ID] = 200;

rem_subscription_data.topic_name = DDS_String_dup("Example HelloWorld");
rem_subscription_data.type_name = DDS_String_dup("HelloWorld");
```

(continues on next page)

(continued from previous page)

```
rem_subscription_data.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

retcode = DPSE_RemoteSubscription_assert(participant,
                                         "Participant_2",
                                         &rem_subscription_data,
                                         HelloWorld_get_key_kind(HelloWorldTypePlugin_
↪get(),
                                         NULL)));
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}
```

### 2.8.2 Writing Samples

Within the generated type support code are declarations of the type-specific *DataWriter*. For the HelloWorld type, this is the HelloWorldDataWriter.

Writing a HelloWorld sample is done by calling the write API of the HelloWorldDataWriter.

```
HelloWorldDataWriter *hw_datawriter;
DDS_ReturnCode_t retcode;
HelloWorld *sample = NULL;

/* Create and set sample */
sample = HelloWorld_create();
if (sample == NULL)
{
    /* failure */
}
sprintf(sample->msg, "Hello World!");

/* Write sample  */
hw_datawriter = HelloWorldDataWriter_narrow(datawriter);

retcode = HelloWorldDataWriter_write(hw_datawriter, sample, &DDS_HANDLE_NIL);
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}
```

For more information, see *Sending Data*.

## 2.9 Create Subscriber

A DDS *Subscriber* contains zero or more *DataReaders*. A publishing application must create at least one DDS *Publisher* and then a *DataReader* for each *Topic* it wants to subscribe to.

Please refer to SubscriberQos and DataReaderQos for customizable policies for these DDS entities.

```
DDS_Subscriber *subscriber = NULL;
subscriber = DDS_DomainParticipant_create_subscriber(participant,
                                                     &DDS_SUBSCRIBER_QOS_DEFAULT,
                                                     NULL,
                                                     DDS_STATUS_MASK_NONE);
if (subscriber == NULL)
{
    /* failure */
}
```

For more information, see *Receiving Data*.

## 2.10 Create DataReader

```
DDS_DataReader *datareader = NULL;
struct DDS_DataReaderQos dr_qos = DDS_DataReaderQos_INITIALIZER;
struct DDS_DataReaderListener dr_listener = DDS_DataReaderListener_INITIALIZER;

/* Configure Reader Qos */

/* Setting the rtps_object_id is only required when using DPSE */
dr_qos.protocol.rtps_object_id = 200;
dr_qos.resource_limits.max_instances = 2;
dr_qos.resource_limits.max_samples_per_instance = 2;
dr_qos.resource_limits.max_samples =
    dr_qos.resource_limits.max_samples_per_instance * dr_qos.resource_limits.max_
↪instances;
dr_qos.reader_resource_limits.max_remote_writers = 10;
dr_qos.reader_resource_limits.max_remote_writers_per_instance = 10;
dr_qos.history.depth = 1;
dr_qos.durability.kind = DDS_VOLATILE_DURABILITY_QOS;
dr_qos.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

/* Set listener callbacks */
dr_listener.on_data_available = HelloWorldSubscriber_on_data_available;
dr_listener.on_subscription_matched = HelloWorldSubscriber_on_subscription_matched;

datareader = DDS_Subscriber_create_datareader(subscriber,
                                              DDS_Topic_as_topicdescription(topic),
                                              &dr_qos,
                                              &dr_listener,
                                              DDS_DATA_AVAILABLE_STATUS | DDS_
↪SUBSCRIPTION_MATCHED_STATUS);
```

(continues on next page)

(continued from previous page)

```
if (datareader == NULL)
{
    /* failure */
}
```

The DataReaderListener has its callbacks selectively enabled by the DDS status mask. In the example, the mask has set the DDS_SUBSCRIPTION_MATCHED_STATUS and DDS_DATA_AVAILABLE_STATUS statuses, and accordingly the DataReaderListener has its on_subscription_matched and on_data_available assigned to callback functions.

```
void HelloWorldSubscriber_on_subscription_matched(void *listener_data,
                                                  DDS_DataReader * reader,
                                                  const struct DDS_
↪SubscriptionMatchedStatus *status)
{
    if (status->current_count_change > 0)
    {
        printf("Matched a publisher\n");
    }
    else
    {
        printf("Unmatched a publisher\n");
    }
}
```

```
void HelloWorldSubscriber_on_data_available(void* listener_data,
                                            DDS_DataReader* reader)
{
    HelloWorldDataReader *hw_reader = HelloWorldDataReader_narrow(reader);
    DDS_ReturnCode_t retcode;
    struct DDS_SampleInfo *sample_info = NULL;
    HelloWorld *sample = NULL;
    struct DDS_SampleInfoSeq info_seq = DDS_SEQUENCE_INITIALIZER;
    struct HelloWorldSeq sample_seq = DDS_SEQUENCE_INITIALIZER;
    const DDS_Long TAKE_MAX_SAMPLES = 32;
    DDS_Long i;

    retcode = HelloWorldDataReader_take(hw_reader,
       &sample_seq, &info_seq, TAKE_MAX_SAMPLES,
       DDS_ANY_SAMPLE_STATE, DDS_ANY_VIEW_STATE, DDS_ANY_INSTANCE_STATE);

    if (retcode != DDS_RETCODE_OK)
    {
        printf("failed to take data: %d\n", retcode);
        goto done;
    }

    /* Print each valid sample taken */
    for (i = 0; i < HelloWorldSeq_get_length(&sample_seq); ++i)
    {
        sample_info = DDS_SampleInfoSeq_get_reference(&info_seq, i);
```

(continues on next page)

(continued from previous page)

```
        if (sample_info->valid_data)
        {
            sample = HelloWorldSeq_get_reference(&sample_seq, i);
            printf("\nSample received\n\tmsg: %s\n", sample->msg);
        }
        else
        {
            printf("not valid data\n");
        }
    }

    HelloWorldDataReader_return_loan(hw_reader, &sample_seq, &info_seq);

 done:
    HelloWorldSeq_finalize(&sample_seq);
    DDS_SampleInfoSeq_finalize(&info_seq);
}
```

### 2.10.1 DPSE Discovery: Assert Remote Publication

A subscribing application using DPSE discovery must specify the other *DataWriters* that its *DataReaders* are allowed to discover. Like the API for asserting a remote participant, the DPSE API for asserting a remote publication must be called for each remote *DataWriter* that a *DataReader* may discover.

```
struct DDS_PublicationBuiltinTopicData rem_publication_data =
    DDS_PublicationBuiltinTopicData_INITIALIZER;

/* Set Writer's protocol.rtps_object_id */
rem_publication_data.key.value[DDS_BUILTIN_TOPIC_KEY_OBJECT_ID] = 100;

rem_publication_data.topic_name = DDS_String_dup("Example HelloWorld");
rem_publication_data.type_name = DDS_String_dup("HelloWorld");

rem_publication_data.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

retcode = DPSE_RemotePublication_assert(participant,
                                        "Participant_1",
                                        &rem_publication_data,
                                        HelloWorld_get_key_kind(HelloWorldTypePlugin_
↪get(),
                                        NULL)));
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}
```

Asserting a remote publication requires configuration of all QoS policies necessary to determine matching.

### 2.10.2 Receiving Samples

Accessing received samples can be done in a few ways:

- **Polling**. Do read or take within a periodic polling loop.
- **Listener**. When a new sample is received, the DataReaderListener's on_data_available is called. Processing is done in the context of the middleware's receive thread. See the above HelloWorldSubscriber_on_data_available callback for example code.
- **Waitset**. Create a waitset, attach it to a status condition with the data_available status enabled, and wait for a received sample to trigger the waitset. Processing is done in the context of the user's application thread, as shown in the code snippet below:

```
DDS_WaitSet *waitset = NULL;
struct DDS_Duration_t wait_timeout = { 10, 0 }; /* 10 seconds */
DDS_StatusCondition *dr_condition = NULL;
struct DDS_ConditionSeq active_conditions = DDS_SEQUENCE_INITIALIZER;

if (!DDS_ConditionSeq_initialize(&active_conditions))
{
    /* failure */
}

if (!DDS_ConditionSeq_set_maximum(&active_conditions, 1))
{
    /* failure */
}

waitset = DDS_WaitSet_new();
if (waitset == NULL )
{
    /* failure */
}

dr_condition = DDS_Entity_get_statuscondition(DDS_DataReader_as_entity(datareader));

retcode = DDS_StatusCondition_set_enabled_statuses(dr_condition,
                                                   DDS_DATA_AVAILABLE_STATUS);
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}

retcode = DDS_WaitSet_attach_condition(waitset,
                                       DDS_StatusCondition_as_condition(dr_condition));
if (retcode != DDS_RETCODE_OK)
{
    /* failure */
}

retcode = DDS_WaitSet_wait(waitset, active_conditions, &wait_timeout);
```

(continues on next page)

(continued from previous page)

```
switch (retcode)
{
    case DDS_RETCODE_OK:
    {
        /* This WaitSet only has a single condition attached to it
         * so we can implicitly assume the DataReader's status condition
         * to be active (with the enabled DATA_AVAILABLE status) upon
         * successful return of wait().
         * If more than one conditions were attached to the WaitSet,
         * the returned sequence must be examined using the
         * commented out code instead of the following.
         */

        HelloWorldSubscriber_take_data(HelloWorldDataReader_narrow(datareader));

        /*
        DDS_Long active_len = DDS_ConditionSeq_get_length(&active_conditions);
        for (i = active_len - 1; i >= 0; --i)
        {
            DDS_Condition *active_condition =
                *DDS_ConditionSeq_get_reference(&active_conditions, i);

            if (active_condition ==
                    DDS_StatusCondition_as_condition(dr_condition))
            {
                total_samples += HelloWorldSubscriber_take_data(
                                HelloWorldDataReader_narrow(datareader));
            }
            else if (active_condition == some_other_condition)
            {
                do_something_else();
            }
        }
        */
        break;
    }
    case DDS_RETCODE_TIMEOUT:
    {
        printf("WaitSet_wait timed out\n");
        break;
    }
    default:
    {
        printf("ERROR in WaitSet_wait: retcode=%d\n", retcode);
        break;
    }
}
```

### 2.10.3 Filtering Samples

*DataReaders* support filtering data in their QoS; see *Content Filtering* for more details. Alternatively, a DataReaderListener in *Connext Micro* provides callbacks in order to perform application-level filtering on a per-sample basis.

- **on_before_sample_deserialize**. Through this callback, a received sample is presented to the application before it has been deserialized or stored in the *DataReader*'s queue.
- **on_before_sample_commit**. Through this callback, a received sample is presented to the application after it has been deserialized but before it has been stored in the *DataReader*'s queue.

You control the callbacks' sample_dropped parameter; upon exiting either callback, the *DataReader* will drop the sample if sample_dropped is true. Consequently, dropped samples are not stored in the *DataReader*'s queue and are not available to be read or taken.

Neither callback is associated with a DDS Status. Rather, each is enabled when assigned, to a non-NULL callback.

NOTE: Because it is called after the sample has been deserialized, on_before_sample_commit provides an additional sample_info parameter, containing some of the usual sample information that would be available when the sample is read or taken.

The HelloWorld_dpde example's subscriber has this on_before_sample_commit callback:

```
DDS_Boolean HelloWorldSubscriber_on_before_sample_commit(
    void *listener_data,
    DDS_DataReader *reader,
    const void *const sample,
    const struct DDS_SampleInfo *const sample_info,
    DDS_Boolean *dropped)
{
    HelloWorld *hw_sample = (HelloWorld *)sample;

    /* Drop samples with even-numbered count in msg */
    HelloWorldSubscriber_filter_sample(hw_sample, dropped);

    if (*dropped)
    {
        printf("\nSample filtered, before commit\n\tDROPPED - msg: %s\n",
               hw_sample->msg);
    }

    return DDS_BOOLEAN_TRUE;
}

...

dr_listener.on_before_sample_commit =
    HelloWorldSubscriber_on_before_sample_commit;
```

For more information, see *Receiving Data*.

# Chapter 3

# Programmer's Guide

## 3.1 Data Types

How data is stored or laid out in memory can vary from language to language, compiler to compiler, operating system to operating system, and processor to processor. This combination of language/compiler/operating system/processor is called a *platform*. Any modern middleware must be able to take data from one specific platform (for example, C/gcc.7.3.0/Linux®/PPC) and transparently deliver it to another (for example, C/gcc.7.3.0/Linux/Arm® v8). This process is commonly called *serialization/deserialization*, or *marshalling/demarshalling*.

*Connext Micro* data samples sent on the same *Connext Micro* topic share a data type. This type defines the fields that exist in the DDS data samples and what their constituent types are. The middleware stores and propagates this meta-information separately from the individual DDS data samples, allowing it to propagate DDS samples efficiently while handling byte ordering and alignment issues for you.

To publish and/or subscribe to data with *Connext Micro*, you will carry out the following steps:

1. Select a type to describe your data and use the *RTI Code Generator* to define a type at compile-time using a language-independent description language.

   The *RTI Code Generator* accepts input in the following formats:

   - `OMG IDL`. This format is a standardized component of the DDS specification. It describes data types with a C++-like syntax. A link to the latest specification can be found here: https://www.omg.org/spec/IDL.
   - `XML in a DDS-specific format`. This XML format is terser, and therefore easier to read and write by hand, than an XSD file. It offers the general benefits of XML-extensibility and ease of integration, while fully supporting DDS-specific data types and concepts. A link to the latest specification, including a description of the XML format, can be found here: https://www.omg.org/spec/DDS-XTypes/.
   - `XSD format`. You can describe data types with XML schemas (XSD). A link to the latest specification, including a description of the XSD format, can be found here: https://www.omg.org/spec/DDS-XTypes/.

2. Register your type with a logical name.
3. Create a *Topic* using the type name you previously registered.

   If you've chosen to use a built-in type instead of defining your own, you will use the API constant corresponding to that type's name.
4. Create one or more *DataWriters* to publish your data and one or more *DataReaders* to subscribe to it.

   The concrete types of these objects depend on the concrete data type you've selected, in order to provide you with a measure of type safety.

Whether publishing or subscribing to data, you will need to know how to create and delete DDS data samples and how to get and set their fields. These tasks are described in Working with DDS Data Samples in the RTI Connext Core Libraries User's Manual.

### 3.1.1 Introduction to the Type System

A *user data type* is any custom type that your application defines for use with *RTI Connext Micro*. It may be a structure, a union, a value type, an enumeration, or a typedef (or language equivalents).

Your application can have any number of user data types, composed of any of the primitive data types listed below (or of other user data types):

- octet, char, wchar
- short, unsigned short
- long, unsigned long
- long long, unsigned long long
- float
- double, long double
- boolean
- enum (with or without explicit values)
- bounded string and wstring

The following type-building constructs are also supported:

- module (also called a package or namespace)
- pointer
- array of primitive or user type elements
- bounded sequence of elements—a sequence is a variable-length ordered collection, such as a vector or list
- typedef
- union

- struct
- value type, a complex type that supports inheritance and other object-oriented features

Only structures, unions, and value types may be read and written directly by *Connext Micro*; enums, typedefs, and primitive types must be contained within a structure, union, or value type. In order for a *DataReader* and *DataWriter* to communicate with each other, the data types associated with their respective Topic definitions must be identical.

To use a data type with *Connext Micro*, you must define that type in a way the middleware understands and then register the type with the middleware. These steps allow *Connext Micro* to serialize, deserialize, and otherwise operate on specific types. They will be described in detail in the following sections.

### Sequences

A sequence contains an ordered collection of elements that are all of the same type. The operations supported in the sequence are documented in the C API Reference and C++ API Reference.

Elements in a sequence are accessed with their index, just like elements in an array. Indices start at zero in all APIs. Unlike arrays, however, sequences can grow in size. A sequence has two sizes associated with it: a physical size (the "maximum") and a logical size (the "length"). The physical size indicates how many elements are currently allocated by the sequence to hold; the logical size indicates how many valid elements the sequence actually holds. The length can vary from zero up to the maximum. Elements cannot be accessed at indices beyond the current length.

A sequence must be declared as bounded. A sequence's "bound" is the maximum number of elements that the sequence can contain at any one time. A finite bound is very important because it allows *RTI Connext Micro* to preallocate buffers to hold serialized and deserialized samples of your types; these buffers are used when communicating with other nodes in your distributed system. The bound can be either explicit or implicit:

- An *explicit* bound is given directly in the IDL:

```
struct MyType
{
    //Maximum of 32 longs
    sequence<long,32> a_long_seq;
}
```

- An *implicit* bound uses the unbounded notation in IDL, but relies on the -sequenceSize parameter passed to *rtiddsgen* for the maximum length:

```
struct MyType
{
    sequence<long> a_long_seq;
}
```

By default, any unbounded sequences found in an IDL file will be given a default bound of 100 elements. This default value can be overwritten using *RTI Code Generator's* `-sequenceSize` command-line argument (see the Command-Line Arguments chapter in the *RTI Code Generator User's Manual*).

### Strings and Wide Strings

*Connext Micro* supports both strings consisting of single-byte characters (the IDL string type) and strings consisting of wide characters (IDL wstring). The wide characters supported by *Connext Micro* are large enough to store two-byte Unicode/UTF16 characters.

Like sequences, strings must be bounded. A string's "bound" is its maximum length (not counting the trailing NULL character in C and C++).

In C and Traditional C++, strings are mapped to `char*`.

By default, any unbounded string found in an IDL file will be given a default bound of 255 elements. This default value can be overwritten using *RTI Code Generator's* `-stringSize` command-line argument (see the Command-Line Arguments chapter in the *RTI Code Generator User's Manual*).

### IDL String Encoding

The "Extensible and Dynamic Topic Types for DDS specification" (https://www.omg.org/spec/DDS-XTypes/) standardizes the default encoding for strings to UTF-8. This encoding shall be used as the wire format. Language bindings may use the representation that is most natural in that particular language. If this representation is different from UTF-8, the language binding shall manage the transformation to/from the UTF-8 wire representation.

As an extension, *Connext Micro* offers ISO_8859_1 as an alternative string wire encoding.

This section describes the encoding for IDL strings across different languages in *Connext Micro* and how to configure that encoding.

- C, Traditional C++

  IDL strings are mapped to a NULL-terminated array of DDS_Char (`char*`). Users are responsible for using the right character encoding (UTF-8 or ISO_8859_1) when populating the string values. This applies to all generated code, DynamicData, and Built-in data types. The middleware does not transform from the language binding encoding to the wire encoding.

### IDL Wide Strings Encoding

The "Extensible and Dynamic Topic Types for DDS specification" (https://www.omg.org/spec/DDS-XTypes/) standardizes the default encoding for wide strings to UTF-16. This encoding shall be used as the wire format.

When the data representation is Extended CDR version 1, wide-string characters have a size of 4 bytes on the wire with UTF-16 encoding. When the data representation is Extended CDR version 2, wide-string characters have a size of 2 bytes on the wire with UTF-16 encoding.

Language bindings may use the representation that is most natural in that particular language. If this representation is different from UTF-16, the language binding shall manage the transformation to/from the UTF-16 wire representation.

- C, Traditional C++

IDL wide strings are mapped to a NULL-terminated array of DDS_Wchar (`DDS_Wchar*`). DDS_Wchar is an unsigned 2-byte integer. Users are responsible for using the right character encoding (UTF-16) when populating the wide-string values. This applies to all generated code, DynamicData, and Built-in data types. *Connext Micro* does not transform from the language binding encoding to the wire encoding.

#### Sending Type Information on the Network

*Connext Professional* can send type information on the network using a concept called type objects. A type object is a description of a type suitable to network transmission, and is commonly used by tools to visualize data from any application.

However, please note that *Connext Micro* does not support sending type information on the network. Instead, tools can load type information from XML files generated from IDL using *rtiddsgen*. Please refer to the Code Generator User's Manual for more information.

#### Extensible Types (XTypes) 1.2 Compatibility

*Connext Micro* supports the "Extensible and Dynamic Topic Types for DDS" (DDS-XTypes) specification from the Object Management Group (OMG), version 1.2 (https://www.omg.org/spec/DDS-XTypes/1.2) with the following limitations:

- Extended Common Data Representation (CDR) encoding version 1 (XCDR) and Extended CDR encoding version 2 (XCDR2) are supported by default.
- If *RTI Code Generator* (*rtiddsgen*) is used with the option `-interpreted 0`, support for XTypes is disabled and only plain CDR is supported (CDRv1 final types).
- *Connext Micro* does not send type information.
- *Connext Micro* does not perform type-compatibility checking based on the type information, only the type-name. This means that advanced XTypes 1.2 features cannot be supported, such as:
  - Type equivalence
  - String-length matching and truncation
  - Sequence-length matching and truncation

### 3.1.2 Creating User Data Types with IDL

You can create user data types in a text file using IDL (Interface Description Language). IDL is programming-language independent, so the same file can be used to generate code in C and Traditional C++. *RTI Code Generator* parses the IDL file and automatically generates all the necessary routines and wrapper functions to bind the types for use by *Connext Micro* at run time. You will end up with a set of required routines and structures that your application and *Connext Micro* will use to manipulate the data.

Please refer to the section on Creating User Data Types with IDL in the RTI Connext DDS Core Libraries User's Manual for more information.

Note: Not all features in *RTI Code Generator* are supported when generating code for *Connext Micro*, see *Unsupported Features of rtiddsgen with Connext Micro*.

### 3.1.3 Working with DDS Data Samples

You should now understand how to define and work with data types. Now that you have chosen one or more data types to work with, this section will help you understand how to create and manipulate objects of those types.

**In C:**

You create and delete your own objects from factories, just as you create *Connext Micro* objects from factories. In the case of user data types, the factory is a singleton object called the type support. Objects allocated from these factories are deeply allocated and fully initialized.

```
/* In the generated header file: */
struct MyData {
    char* myString;
};
/* In your code: */
MyData* sample = MyDataTypeSupport_create_data();
char* str = sample->myString; /*empty, non-NULL string*/
/* ... */
MyDataTypeSupport_delete_data(sample);
```

**In Traditional C++:**

Without the `-constructor option`, you create and delete objects using the TypeSupport factories.

```
MyData* sample = MyDataTypeSupport::create_data();
char* str = sample->myString; // empty, non-NULL string
// ...
MyDataTypeSupport::delete_data(sample);
```

Please refer to the section on Working with DDS Data Samples in the RTI Connext DDS Core Libraries User's Manual for more information.

## 3.2 DDS Entities

The main classes extend an abstract base class called a DDS *Entity*. Every DDS *Entity* has a set of associated events known as statuses and a set of associated Quality of Service policies (QoS policies). In addition, a *Listener* may be registered with the *Entity* to be called when status changes occur. DDS *Entities* may also have attached DDS *Conditions*, which provide a way to wait for status changes. Figure 3.1 presents an overview in a UML diagram.

Please note that *RTI Connext Micro* does not support the following:

- **MultiTopic**

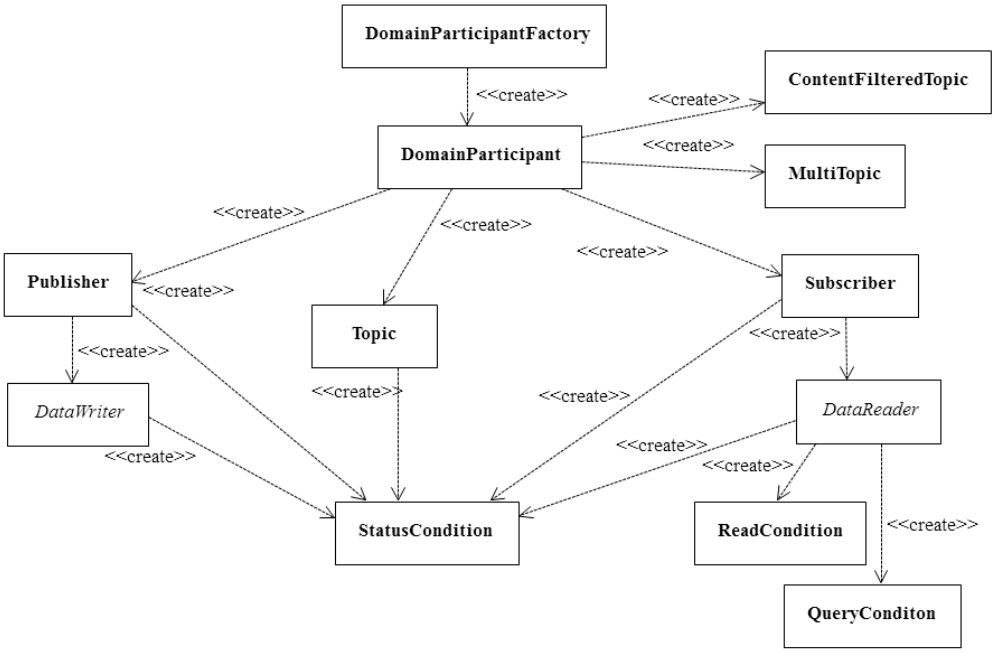



Figure 3.1: Overview of DDS Entities

- **ContentFilteredTopic**[1]
- **ReadCondition**
- **QueryConditions**

For a general description of DDS *Entities* and their operations, please refer to the DDS Entities chapter in the RTI Connext DDS Core Libraries User's Manual. Note that *RTI Connext Micro* does not support all APIs and QosPolicies; please refer to the C API Reference and C++ API Reference documentation for more information.

## 3.3 Sending Data

This section discusses how to create, configure, and use *Publishers* and *DataWriters* to send data. It describes how these *Entities* interact, as well as the types of operations that are available for them.

The goal of this section is to help you become familiar with the *Entities* you need for sending data. For up-to-date details such as formal parameters and return codes on any mentioned operations, please see the C API Reference and C++ API Reference documentation.

[1] **ContentFilteredTopics** are not supported, but *Connext Micro* does support filtering data with a QoS policy. See *Content Filtering* for more information.

### 3.3.1 Preview: Steps to Sending Data

To send DDS samples of a data instance:

1. Create and configure the required *Entities*:
   a. Create a *DomainParticipant*.
   b. Register user data types with the *DomainParticipant*. For example, the `FooDataType`.
   c. Use the *DomainParticipant* to create a *Topic* with the registered data type.
   d. Use the *DomainParticipant* to create a *Publisher*.
   e. Use the *Publisher* or *DomainParticipant* to create a *DataWriter* for the *Topic*.
   f. Use a type-safe method to cast the generic *DataWriter* created by the *Publisher* to a type-specific *DataWriter*. For example, `FooDataWriter`. Optionally, register data instances with the *DataWriter*. If the *Topic*'s user data type contain key fields, then registering a data instance (data with a specific key value) will improve performance when repeatedly sending data with the same key. You may register many different data instances; each registration will return an instance handle corresponding to the specific key value. For non-keyed data types, instance registration has no effect.
2. Every time there is changed data to be published:
   a. Store the data in a variable of the correct data type (for instance, variable `Foo` of the type `FooDataType`).
   b. Call the `FooDataWriter`'s `write()` operation, passing it a reference to the variable `Foo`.
      - For non-keyed data types or for non-registered instances, also pass in `DDS_HANDLE_NIL`.
      - For keyed data types, pass in the instance handle corresponding to the instance stored in `Foo`, if you have registered the instance previously. This means that the data stored in `Foo` has the same key value that was used to create instance handle.
   c. The `write()` function will take a snapshot of the contents of `Foo` and store it in *Connext Micro* internal buffers from where the DDS data sample is sent under the criteria set by the *Publisher's* and *DataWriter's* QoS policies.

### 3.3.2 Publishers

An application that intends to publish information needs the following *Entities*: *DomainParticipant*, *Topic*, *Publisher*, and *DataWriter*. All *Entities* have a corresponding specialized *Listener* and a set of QoS policies. A *Listener* is how *Connext Micro* notifies your application of status changes relevant to the *Entity*. The QoS policies allow your application to configure the behavior and resources of the *Entity*.

- A *DomainParticipant* defines the DDS domain in which the information will be made available.

- A *Topic* defines the name under which the data will be published, as well as the type (format) of the data itself.
- An application writes data using a *DataWriter*. The *DataWriter* is bound at creation time to a *Topic*, thus specifying the name under which the *DataWriter* will publish the data and the type associated with the data. The application uses the *DataWriter's* `write()` operation to indicate that a new value of the data is available for dissemination.
- A *Publisher* manages the activities of several *DataWriters*. The *Publisher* determines when the data is actually sent to other applications. Depending on the settings of various QoS policies of the *Publisher* and *DataWriter*, data may be buffered to be sent with the data of other *DataWriters* or not sent at all.

  You may have multiple *Publishers*, each managing a different set of *DataWriters*, or you may choose to use one *Publisher* for all your *DataWriters*.

### 3.3.3 DataWriters

To create a *DataWriter*, you need a *DomainParticipant*, *Publisher*, and a *Topic*.

You need a *DataWriter* for each *Topic* that you want to publish. For more details on all operations, see the C API Reference and C++ API Reference documentation.

For more details on creating, deleting, and setting up *DataWriters*, see replace:: the DataWriters section in the RTI Connext DDS Core Libraries User's Manual.

### 3.3.4 Supported QoS policies

Please refer to the C API Reference and C++ API Reference for details on QoS policies supported by *DataWriters*, *Publishers*, and *Subscribers*.

## 3.4 Receiving Data

This section discusses how to create, configure, and use *Subscribers* and *DataReaders* to receive data. It describes how these objects interact, as well as the types of operations that are available for them.

The goal of this section is to help you become familiar with the *Entities* you need for receiving data. For up-to-date details such as formal parameters and return codes on any mentioned operations, please see the C API Reference and C++ API Reference documentation.

**Warning:** *Connext Micro DataReaders* cannot match with or receive data from *Connext DataWriters* that are configured to send compressed data. See *Interoperating with Connext Professional Discovery* for more information.

### 3.4.1 Preview: Steps to Receiving Data

There are three ways to receive data:

- Your application can explicitly check for new data by calling a *DataReader's* `read()` or `take()` operation. This method is also known as *polling for data*.
- Your application can be notified whenever new DDS data samples arrive—this is done with a *Listener* on either the *Subscriber* or the *DataReader*. *RTI Connext Micro* will invoke the *Listener's* callback routine when there is new data. Within the callback routine, user code can access the data by calling `read()` or `take()` on the *DataReader*. This method is the way for your application to receive data with the least amount of latency.
- Your application can wait for new data by using *Conditions* and a *WaitSet*, then calling `wait()`. *Connext Micro* will block your application's thread until the criteria (such as the arrival of DDS samples, or a specific status) set in the *Condition* becomes true. Then your application resumes and can access the data with `read()` or `take()`.

The *DataReader*'s `read()` and `take()` operations will both return the data to your application, but they treat the returned data differently. The `take()` operation removes the data from the *DataReader*'s receive queue, so it is no longer stored by *Connext Micro* and cannot be read again. The `read()` operation continues to store the data in the *DataReader*'s receive queue, where it *can* be read again.

To prepare to receive data, create and configure the required Entities:

1. Create a *DomainParticipant*.
2. Register user data types with the *DomainParticipant*. For example, the `FooDataType`.
3. Use the *DomainParticipant* to create a *Topic* with the registered data type.
4. Use the *DomainParticipant* to create a *Subscriber*.
5. Use the *Subscriber* or *DomainParticipant* to create a *DataReader* for the *Topic*.
6. Use a type-safe method to cast the generic *DataReader* created by the *Subscriber* to a type-specific *DataReader*. For example, `FooDataReader`.

Then use one of the following mechanisms to receive data.

- To receive DDS data samples by polling for new data:

  Using a `FooDataReader`, use the `read()` or `take()` operations to access the DDS data samples that have been received and stored for the *DataReader*. These operations can be invoked at any time, even if the receive queue is empty.

- To receive DDS data samples when new samples arrive:

  Install a *Listener* on the *DataReader* or *Subscriber* that will be called back by an internal *Connext Micro* thread when new DDS data samples arrive for the *DataReader*.

  1. Create a *DDSDataReaderListener* for the *FooDataReader* or a *DDSSubscriberListener* for *Subscriber*. In C++ you must derive your own *Listener* class from those base classes. In C, you must create the individual functions and store them in a structure.

If you created a *DDSDataReaderListener* with the `on_data_available()` callback enabled: `on_data_available()` will be called when new data arrives for that `DataReader`.

If you created a *DDSSubscriberListener* with the `on_data_on_readers()` callback enabled: `on_data_on_readers()` will be called when data arrives for any *DataReader* created by the *Subscriber*.

2. Install the *Listener* on either the *FooDataReader* or *Subscriber*.

   For the *DataReader*, the *Listener* should be installed to handle changes in the `DATA_AVAILABLE` status.

   For the *Subscriber*, the *Listener* should be installed to handle changes in the `DATA_ON_READERS` status.

3. Only 1 *Listener* will be called back when new data arrives for a *DataReader*.

   *Connext Micro* will call the *Subscriber's Listener* if it is installed. Otherwise, the *DataReader's Listener* is called if it is installed. That is, the `on_data_on_readers()` operation takes precedence over the `on_data_available()` operation.

   If neither *Listeners* are installed or neither *Listeners* are enabled to handle their respective statuses, then *Connext Micro* will not call any user functions when new data arrives for the *DataReader*.

4. In the `on_data_available()` method of the *DDSDataReaderListener*, invoke `read()` or `take()` on the *FooDataReader* to access the data.

   If the `on_data_on_readers()` method of the *DDSSubscriberListener* is called, the code can invoke `read()` or `take()` directly on the *Subscriber's DataReaders* that have received new data. Alternatively, the code can invoke the *Subscriber's* `notify_datareaders()` operation. This will in turn call the `on_data_available()` methods of the *DataReaderListeners* (if installed and enabled) for each of the *DataReaders* that have received new DDS data samples.

- To wait (block) until DDS data samples arrive:

  1. Use the *DataReader* to create a *StatusCondition* that describes the DDS samples for which you want to wait. For example, you can specify that you want to wait for never-before-seen DDS samples from *DataReaders* that are still considered to be 'alive.'
  2. Create a *WaitSet*.
  3. Attach the *StatusCondition* to the *WaitSet*.
  4. Call the *WaitSet's* `wait()` operation, specifying how long you are willing to wait for the desired DDS samples. When `wait()` returns, it will indicate that it timed out, or that the attached Condition become true (and therefore the desired DDS samples are available).
  5. Using a `FooDataReader`, use the `read()` or `take()` operations to access the DDS data samples that have been received and stored for the *DataReader*.

### 3.4.2 Subscribers

An application that intends to subscribe to information needs the following *Entities*: *DomainParticipant*, *Topic*, *Subscriber*, and *DataReader*. All *Entities* have a corresponding specialized *Listener* and a set of QoS policies. The *Listener* is how *RTI Connext Micro* notifies your application of status changes relevant to the *Entity*. The QoS policies allow your application to configure the behavior and resources of the *Entity*.

- The *DomainParticipant* defines the DDS domain on which the information will be available.
- The *Topic* defines the name of the data to be subscribed, as well as the type (format) of the data itself.
- The *DataReader* is the *Entity* used by the application to subscribe to updated values of the data. The *DataReader* is bound at creation time to a *Topic*, thus specifying the named and typed data stream to which it is subscribed. The application uses the *DataWriter*'s `read()` or `take()` operation to access DDS data samples received for the *Topic*.
- The *Subscriber* manages the activities of several *DataReader* entities. The application receives data using a *DataReader* that belongs to a *Subscriber*. However, the *Subscriber* will determine when the data received from applications is actually available for access through the *DataReader*. Depending on the settings of various QoS policies of the *Subscriber* and *DataReader*, data may be buffered until DDS data samples for associated *DataReaders* are also received. By default, the data is available to the application as soon as it is received.

For more information on creating and deleting *Subscribers*, as well as setting QoS policies, see the Subscribers section in the RTI Connext DDS Core Libraries User's Manual.

### 3.4.3 DataReaders

To create a *DataReader*, you need a *DomainParticipant*, a *Topic*, and a *Subscriber*. You need at least one *DataReader* for each *Topic* whose DDS data samples you want to receive.

For more details on all operations, see the C API Reference and C++ API Reference.

### 3.4.4 Using DataReaders to Access Data (Read & Take)

For user applications to access the data received for a *DataReader*, they must use the type-specific derived class or set of functions in the C API Reference. Thus, for a user data type `Foo`, you must use methods of the `FooDataReader` class. The type-specific class or functions are automatically generated if you use *RTI Code Generator*.

### 3.4.5 Supported QoS policies

Please refer to the C API Reference and C++ API Reference for details on supported Qos Policies for *Subscribers* and *DataReaders*.

## 3.5 DDS Domains

This section discusses how to use *DomainParticipants*. It describes the types of operations that are available for them and their QosPolicies.

The goal of this section is to help you become familiar with the objects you need for setting up your *RTI Connext Micro* application. For specific details on any mentioned operations, see the C API Reference and C++ API Reference documentation.

### 3.5.1 Fundamentals of DDS Domains and DomainParticipants

*DomainParticipants* are the focal point for creating, destroying, and managing other *RTI Connext Micro* objects. A DDS *domain* is a logical network of applications: only applications that belong to the same DDS *domain* may communicate using *Connext Micro*. A DDS *domain* is identified by a unique integer value known as a domain ID. An application participates in a DDS domain by creating a *DomainParticipant* for that domain ID.

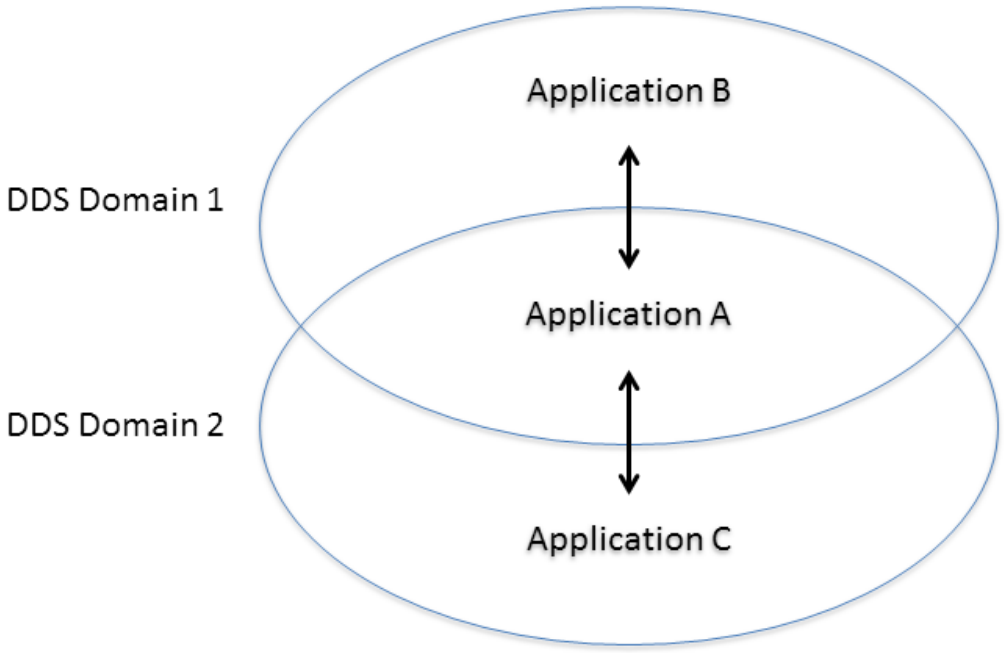



Figure 3.2: Relationship between Applications and DDS Domains—Applications can belong to multiple DDS domains. *A* belongs to DDS domains 1 and 2. Applications in the same DDS domain can communicate with each other, such as *A* and *B*, or *A* and *C*. Applications in different DDS domains, such as *B* and *C*, are not aware of each other and will not exchange messages.

As seen in Figure 3.2, a single application can participate in multiple DDS domains by creating multiple *DomainParticipants* with different domain IDs. *DomainParticipants* in the same DDS domain form a logical network; they are isolated from *DomainParticipants* of other DDS domains, even those running on the same set of physical computers sharing the same physical network. *DomainParticipants* in different DDS domains will never exchange messages with each other. Thus, a DDS domain establishes a "virtual network" linking all *DomainParticipants* that share the same domain ID.

An application that wants to participate in a certain DDS domain will need to create a *DomainParticipant*. As seen in Figure 3.3, a *DomainParticipant* object is a container for all other *Entities* that belong to the same DDS domain. It acts as factory for the *Publisher*, *Subscriber*, and *Topic* entities. (As seen in *Sending Data* and *Receiving Data*, in turn, *Publishers* are factories for *DataWriters* and *Subscribers* are factories for *DataReaders*). *DomainParticipants* cannot contain other *DomainParticipants*.

Like all *Entities*, *DomainParticipants* have QosPolicies and *Listeners*. The *DomainParticipant* entity also allows you to set 'default' values for the QosPolicies for all the entities created from it or from the entities that it creates (*Publishers*, *Subscribers*, *Topics*, *DataWriters*, and *DataReaders*).

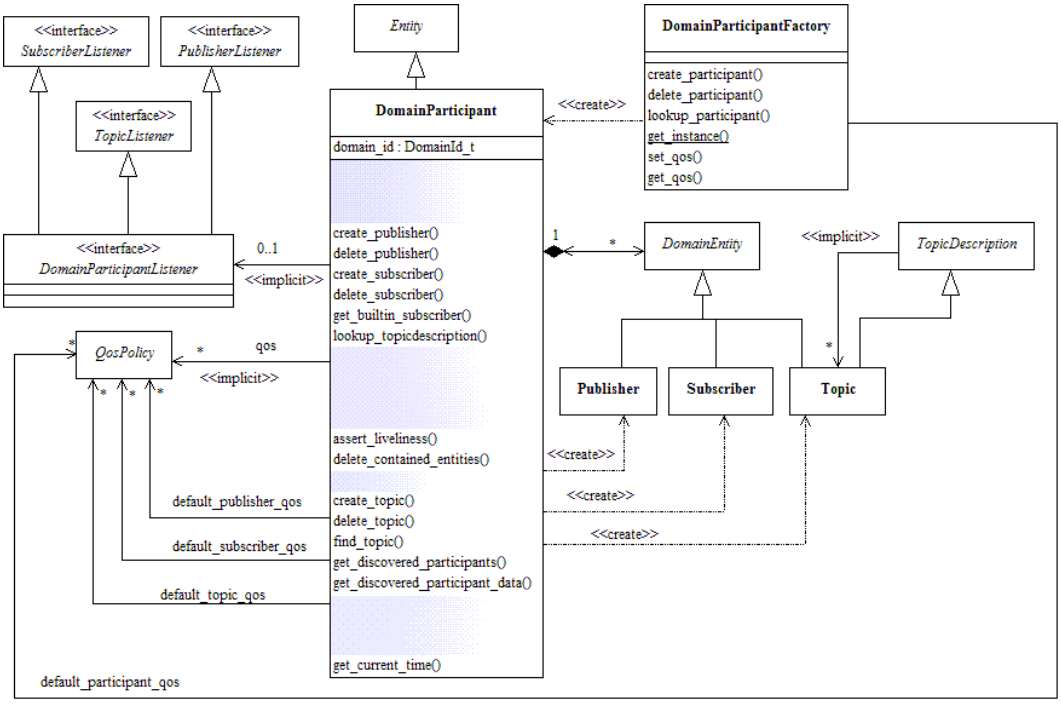



Figure 3.3: DDS Domain Module—Unsupported operations in *Connext Micro* are blurred out.

### 3.5.2 Discovery Announcements

Each *DomainParticipant* announces locators that other *DomainParticipants* must use to communicate with it. A locator is an address that consists of an address kind, a port number, and an address. Four locator types are defined:

- A **unicast meta-traffic locator**. This locator type is used to identify where unicast discovery messages shall be sent. A maximum of four locators of this type can be specified.
- A **multicast meta-traffic locator**. This locator type is used to identify where multicast discovery messages shall be sent. A maximum of four locators of this type can be specified.
- A **unicast user-traffic locator**. This locator type is used to identify where unicast user-traffic messages shall be sent. A maximum of four locators of this type can be specified.
- A **multicast user-traffic locator**. This locator type is used to identify where multicast user-traffic messages shall be sent. A maximum of four locators of this type can be specified.

It is important to note that a maximum of *four* locators of *each* kind can be sent in a *DomainParticipant* discovery message.

The locators in a *DomainParticipant*'s discovery announcement is used for two purposes:

- It informs other *DomainParticipants* where to send their discovery announcements to this *DomainParticipants*.
- It informs the *DataReaders* and *DataWriters* in other *DomainParticipants* where to send data to the *DataReaders* and *DataWriters* in this *DomainParticipant* unless a *DataReader* or *DataWriter* specifies its own locators.

If a *DataReader* or *DataWriter* specifies its own locators, only user-traffic locators can be specified, and the exact same rules apply as for a *DomainParticipant*.

This document uses *address* and *locator* interchangeably. An address corresponds to the port and address part of a locator. The same address may exist as different kinds, in which case they are unique.

For more details about the discovery process, see the *Discovery* section.

## 3.6 Discovery

This section discusses the implementation of discovery plugins in *RTI Connext Micro*. For a general overview of discovery in *RTI Connext Micro*, see *What is Discovery?*.

*Connext Micro* discovery traffic is conducted through transports. Please see the *Transports* section for more information about registering and configuring transports.

### 3.6.1 What is Discovery?

Discovery is the behind-the-scenes way in which *RTI Connext Micro* objects (*DomainParticipants*, *DataWriters*, and *DataReaders*) on different nodes find out about each other. Each *DomainParticipant* maintains a database of information about all the active *DataReaders* and *DataWriters* that are in the same DDS domain. This database is what makes it possible for *DataWriters* and *DataReaders* to communicate. To create and refresh the database, each application follows a common discovery process.

This section describes the default discovery mechanism known as the Simple Discovery Protocol, which includes two phases: *Simple Participant Discovery* and *Simple Endpoint Discovery*.

The goal of these two phases is to build, for each *DomainParticipant*, a complete picture of all the entities that belong to the remote participants that are in its peers list. The peers list is the list of nodes with which a participant may communicate. It starts out the same as the *initial_peers* list that you configure in the DISCOVERY QosPolicy. If the accept_unknown_peers flag in that same QosPolicy is TRUE, then other nodes may also be added as they are discovered; if it is FALSE, then the peers list will match the initial_peers list, plus any peers added using the *DomainParticipant's* `add_peer()` operation.

The following section discusses how *Connext Micro* objects on different nodes find out about each other using the default Simple Discovery Protocol (SDP). It describes the sequence of messages that are passed between *Connext Micro* on the sending and receiving sides.

The discovery process occurs automatically, so you do not have to implement any special code. For more information about advanced topics related to Discovery, please refer to the Discovery chapter in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

#### Simple Participant Discovery

This phase of the Simple Discovery Protocol is performed by the Simple Participant Discovery Protocol (SPDP).

During the Participant Discovery phase, *DomainParticipants* learn about each other. The *DomainParticipant*'s details are communicated to all other *DomainParticipants* in the same DDS domain by sending participant declaration messages, also known as participant *DATA* submessages. The details include the *DomainParticipant*'s unique identifying key (GUID or Globally Unique ID described below), transport locators (addresses and port numbers), and QoS. These messages are sent on a periodic basis using best-effort communication.

*Participant DATAs* are sent periodically to maintain the liveliness of the *DomainParticipant*. They are also used to communicate changes in the *DomainParticipant*'s QoS. Only changes to QosPolicies that are part of the *DomainParticipant*'s built-in data need to be propagated.

When receiving remote participant discovery information, *RTI Connext Micro* determines if the local participant matches the remote one. A 'match' between the local and remote participant occurs only if the local and remote participant have the same Domain ID and Domain Tag. This matching process occurs as soon as the local participant receives discovery information from the remote one. If there is no match, the discovery DATA is ignored, resulting in the remote participant (and all its associated entities) not being discovered.

When a *DomainParticipant* is deleted, a participant *DATA (delete)* submessage with the *DomainParticipant*'s identifying GUID is sent.

The GUID is a unique reference to an entity. It is composed of a GUID prefix and an Entity ID. By default, the GUID prefix is calculated from the IP address and the process ID. The entityID is set by *Connext Micro* (you may be able to change it in a future version).

Once a pair of remote participants have discovered each other, they can move on to the Endpoint Discovery phase, which is how *DataWriters* and *DataReaders* find each other.

### Simple Endpoint Discovery

This phase of the Simple Discovery Protocol is performed by the Simple Endpoint Discovery Protocol (SEDP).

During the Endpoint Discovery phase, *RTI Connext Micro* matches *DataWriters* and *DataReaders*. Information (GUID, QoS, etc.) about your application's *DataReaders* and *DataWriters* is exchanged by sending publication/subscription declarations in DATA messages that we will refer to as *publication DATAs* and *subscription DATAs*. The Endpoint Discovery phase uses reliable communication.

These declaration or DATA messages are exchanged until each *DomainParticipant* has a complete database of information about the participants in its peers list and their entities. Then the discovery process is complete and the system switches to a steady state. During steady state, *participant DATAs* are still sent periodically to maintain the liveliness status of participants. They may also be sent to communicate QoS changes or the deletion of a *DomainParticipant*.

When a remote *DataWriter/DataReader* is discovered, *Connext Micro* determines if the local application has a matching *DataReader/DataWriter*. A 'match' between the local and remote entities occurs only if the *DataReader* and *DataWriter* have the same *Topic*, same data type, and compatible QosPolicies. Furthermore, if the *DomainParticipant* has been set up to ignore certain *DataWriters/DataReaders*, those entities will not be considered during the matching process.

This 'matching' process occurs as soon as a remote entity is discovered, even if the entire database is not yet complete: that is, the application may still be discovering other remote entities.

A *DataReader* and *DataWriter* can only communicate with each other if each one's application has hooked up its local entity with the matching remote entity. That is, both sides must agree to the connection.

Please refer to the section on Discovery Implementation in the RTI Connext DDS Core Libraries User's Manual for more details about the discovery process (available here if you have Internet access).

### 3.6.2 Configuring Participant Discovery Peers

An *RTI Connext Micro DomainParticipant* must be able to send participant discovery announcement messages for other *DomainParticipants* to discover itself, and it must receive announcements from other *DomainParticipants* to discover them.

To do so, each *DomainParticipant* will send its discovery announcements to a set of locators known as its peer list, where a peer is the transport locator of one or more potential other *DomainParticipants* to discover.

#### The Peer Descriptor

A peer descriptor string of the initial_peers sequence defines the interface and address of the locator to which to send, as well as the indices of participants to which to send. The peer descriptor format is:

```
< > denotes optional
[ ] denotes range or discreet values, unless enclosed in ''
    which means a literal.

PEER_DESCRIPTOR = <INDEX@>TRANSPORT_NAME://<ADDRESS>

TRANSPORT_NAME = [a-zA-Z_][0-9a-zA-Z_]+

ADDRESS = 0 or more 8bit characters

INDEX = INTEGER | '[' INTEGER ']' | '[' INTEGER-INTEGER ']' | '[' -INTEGER ']'

INTEGER = DEC_INTEGER | HEX_INTEGER

DEC_INTEGER = [0-9]+

HEX_INTEGER = [0x|0X][0-9a-fA-F]+
```

The `TRANSPORT_NAME` prefix refers to a registered transport name, such as `_udp` or `_shmem` for *Connext Micro*'s builtin transports, or a custom name for a user-defined transport.

`ADDRESS` is a string that you must format as the transport requires; for example, *Connext Micro*'s builtin UDP transport requires an IPv4 address formatted as `w.x.y.z`, while the SHMEM transport omits the `ADDRESS` string entirely. For user-created transports, format the `ADDRESS` as needed.

`INDEX` sets the maximum number of *DomainParticipants* that will be contacted at an address for discovery. If `INDEX` is left blank, *Connext Micro* will use an implied index value of `4@`. For example, `_shmem://` will parse as `4@_shmem://` and will try to connect to the first 5 *DomainParticipants* on the same host via the SHMEM transport. `9@_udp://192.168.0.1` will use the `_udp` transport to try to connect to the first 10 *DomainParticipants* at the IP address `192.168.0.1`.

Remember that every *DomainParticipant* has a participant index that is unique within a DDS domain. The participant index (also referred to as the participant ID), together with the DDS domain ID, is used to calculate the network port on which *DataReaders* of that participant will receive messages. Thus, by specifying the participant index, or a range of indices, for a peer locator,

that locator becomes a port to which messages will be sent only if addressed to the entities of a particular *DomainParticipant*. Specifying indices restricts the number of participant announcements sent to a locator where other *DomainParticipants* exist and, thus, should be considered to minimize network bandwidth usage.

For example:

```
DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers, 5);
DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers, 5);

/* If the index is not specified, it defaults to 5, thus sending to
 * the first 6 participant IDs at IP address 192.168.1.1 using
 * the transport registered as _udp.
 */
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 0) =
     DDS_String_dup("_udp://192.168.1.1");

/* Only send participant annoucements to multicast address 239.255.0.1
 * using the transport registered as _udp. Note that for multicast
 * addresses the index is not relevant since it is a shared address.
 */
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 1) =
     DDS_String_dup("_udp://239.255.0.1");

/* Send annoucements to participant ID 1,2,3, and 4 on 10.10.30.101
 * using the transport registered as _udp.
 */
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 2) =
     DDS_String_dup("[1-4]@_udp://10.10.30.101");

/* Send annoucements to participant ID 2 on address 10.10.30.102
 * using the transport registered as _udp.
 */
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 3) =
     DDS_String_dup("[2]@_udp://10.10.30.102");

/* Send annoucements to participant ID 0-8 on address 10.10.30.102
 * using the transport registered as _udp.
 */
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 4) =
     DDS_String_dup("8@_udp://10.10.30.102");
```

### 3.6.3 Configuring Initial Peers and Adding Peers

DiscoveryQosPolicy_initial_peers is the list of peers a *DomainParticipant* sends its participant announcement messages, when it is enabled, as part of the discovery process.

DiscoveryQosPolicy_initial_peers is an empty sequence by default.

Peers can also be added to the list, before and after a *DomainParticipant* has been enabled, by using DomainParticipant_add_peer.

The *DomainParticipant* will start sending participant announcement messages to the new peer as soon as it is enabled.

### 3.6.4 Configuring Discovery Data Reception

In order to *receive* discovery and user data, it is necessary to configure the `DomainParticipantQos.discovery.enabled_transports` sequence. This is a sequence of transport addresses to listen for discovery data on, and is sent as part of the participant annoucements. Other *DomainParticipants* will send to these addresses.

The address format for configuring data reception uses the following format:

```
< > denotes optional
[ ] denotes range or discreet values, unless enclosed in ''
    which means a literal.

ENABLED_TRANSPORTS = TRANSPORT_NAME://<ADDRESS>

TRANSPORT_NAME = [a-zA-Z_][0-9a-zA-Z_]+

ADDRESS = 0 or more 8bit characters
```

The `TRANSPORT_NAME` prefix and `ADDRESS` are functionally identical to those in *The Peer Descriptor*.

For example, to receive on a single unicast address:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.discovery.enabled_transports, 1);
DDS_StringSeq_set_length(&DomainParticipantQos.discovery.enabled_transports, 1);

/* Receive on the unicast address 192.168.1.1 using the transport registered
 * as _udp.
 */
*DDS_StringSeq_get_reference(&DomainParticipantQos.discovery.enabled_transports, 0) =
                              DDS_String_dup("_udp://192.168.1.1");
```

To receive on all unicast addresses allowed by the transport:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.discovery.enabled_transports, 1);
DDS_StringSeq_set_length(&DomainParticipantQos.discovery.enabled_transports, 1);

/* Receive on all unicast addresses allowed by the transport registered
 * as _udp. This is not recommended if more than 4 network interfaces are
 * allowed as it is non-deterministic which interfaces will be used.
 */
*DDS_StringSeq_get_reference(&DomainParticipantQos.discovery.enabled_transports, 0) =
                              DDS_String_dup("_udp://");
```

To receive on one unicast address and one multicast address:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.discovery.enabled_transports, 2);
DDS_StringSeq_set_length(&DomainParticipantQos.discovery.enabled_transports, 2);
```

(continues on next page)

(continued from previous page)

```
*DDS_StringSeq_get_reference(&DomainParticipantQos.discovery.enabled_transports, 0) =
                              DDS_String_dup("_udp://192.168.1.1");

*DDS_StringSeq_get_reference(&DomainParticipantQos.discovery.enabled_transports, 1) =
                              DDS_String_dup("_udp://239.255.0.1");
```

To receive on one multicast address:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.discovery.enabled_transports, 1);
DDS_StringSeq_set_length(&DomainParticipantQos.discovery.enabled_transports, 1);

*DDS_StringSeq_get_reference(&DomainParticipantQos.discovery.enabled_transports, 0) =
                              DDS_String_dup("_udp://239.255.0.1");
```

### 3.6.5 Configuring User Data Reception

In order to *receive* user data, you must configure the `DomainParticipantQos.user_traffic.enabled_transports` sequence. This is a sequence of default transport addresses to listen for user data on, unless a *DataReader* or *DataWriter* specifies its own address, and is sent as part of the participant annoucements. Other *DomainParticipants* will send to these addresses.

The address format for configuring data reception uses the following format:

```
< > denotes optional
[ ] denotes range or discreet values, unless enclosed in ''
    which means a literal.

ENABLED_TRANSPORTS = TRANSPORT_NAME://<ADDRESS>

TRANSPORT_NAME = [a-zA-Z_][0-9a-zA-Z_]+

ADDRESS = 0 or more 8bit characters
```

The `TRANSPORT_NAME` prefix and `ADDRESS` are functionally identical to those in *The Peer Descriptor*.

For example, to receive on a single unicast address:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.user_traffic.enabled_transports, 1);
DDS_StringSeq_set_length(&DomainParticipantQos.user_traffic.enabled_transports, 1);

/* Receive on the unicast address 192.168.1.1 using the transport registered
 * as _udp.
 */
*DDS_StringSeq_get_reference(&DomainParticipantQos.user_traffic.enabled_transports, 0) =
                              DDS_String_dup("_udp://192.168.1.1");
```

To receive on all unicast addresses allowed by the transport:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.user_traffic.enabled_transports, 1);
DDS_StringSeq_set_length(&DomainParticipantQos.user_traffic.enabled_transports, 1);

/* Receive on all unicast addresses allowed by the transport registered
 * as _udp. This is not recommended if more than 4 network interfaces are
 * allowed as it is non-deterministic which interfaces will be used.
 */
*DDS_StringSeq_get_reference(&DomainParticipantQos.user_traffic.enabled_transports, 0) =
                              DDS_String_dup("_udp://");
```

To receive on one unicast address and one multicast address:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.user_traffic.enabled_transports, 2);
DDS_StringSeq_set_length(&DomainParticipantQos.user_traffic.enabled_transports, 2);

*DDS_StringSeq_get_reference(&DomainParticipantQos.user_traffic.enabled_transports, 0) =
                              DDS_String_dup("_udp://192.168.1.1");

*DDS_StringSeq_get_reference(&DomainParticipantQos.user_traffic.enabled_transports, 1) =
                              DDS_String_dup("_udp://239.255.0.1");
```

---

**Note:** When both multicast and unicast is specified, the following rules are used:

- New data is sent over multicast.
- Retransmissions are sent over unicast.

---

To receive on one multicast address:

```
DDS_StringSeq_set_maximum(&DomainParticipantQos.user_traffic.enabled_transports, 1);
DDS_StringSeq_set_length(&DomainParticipantQos.user_traffic.enabled_transports, 1);

*DDS_StringSeq_get_reference(&DomainParticipantQos.user_traffic.enabled_transports, 0) =
                              DDS_String_dup("_udp://239.255.0.1");
```

### 3.6.6 Configuring User Data Reception per DataReader or DataWriter

A *DataReader* and *DataWriter* can specify their own addresses in the `DataReaderQos.transport.enabled_transports` and `DataWriterQos.transport.enabled_transports` policies. The address format is exactly the same as for `DomainParticipantQos.user_traffic.enabled_transports`, with the restriction that a *DataWriter* can only specify its own unicast addresses.

### 3.6.7 Discovery Plugins

When a *DomainParticipant* receives a participant discovery message from another *DomainParticipant*, it will engage in the process of exchanging information of user-created *DataWriter* and *DataReader* endpoints.

*RTI Connext Micro* provides two ways of determinig endpoint information of other *DomainParticipants*: *Dynamic Discovery Plugin* and *Static Discovery Plugin*.

#### Dynamic Discovery Plugin

Dynamic endpoint discovery uses builtin discovery *DataWriters* and *DataReader* to exchange messages about user created *DataWriter* and *DataReaders*. A *DomainParticipant* using dynamic participant, dynamic endpoint (DPDE) discovery will have a pair of builtin *DataWriters* for sending messages about its own user created *DataWriters* and *DataReaders*, and a pair of builtin *DataReaders* for receiving messages from other *DomainParticipants* about their user created *DataWriters* and *DataReaders*.

Given a *DomainParticipant* with a user *DataWriter*, receiving an endpoint discovery message for a user *DataReader* allows the *DomainParticipant* to get the type, topic, and QoS of the *DataReader* that determine whether the *DataReader* is a match. When a matching *DataReader* is discovered, the *DataWriter* will include that *DataReader* and its locators as destinations for its subsequent writes.

---

**Note:** *RTI Connext* uses the acronyms SPDP and SEDP to distinguish between the two phases of Simple Discovery: participant and endpoint phases (see Discovery in the Core Libraries User's Manual). *RTI Connext Micro* uses the acronyms DPSE and DPDE to distinguish between the static and dynamic endpoint discovery plugins available in *RTI Connext Micro*. The DPSE plugin implements the SPDP protocol and DPDE implements the SPDP and SEDP protocols.

---

### Static Discovery Plugin

Static endpoint discovery uses function calls to statically assert information about remote endpoints belonging to remote *DomainParticipants*. An application with a *DomainParticipant* using dynamic participant, static endpoint (DPSE) discovery has control over which endpoints belonging to particular remote *DomainParticipants* are discoverable.

Whereas dynamic endpoint-discovery can establish matches for all endpoint-discovery messages it receives, static endpoint-discovery establishes matches only for the endpoint that have been asserted programmatically.

With DPSE, a user needs to know *a priori* the configuration of the entities that will need to be discovered by its application. The user must know the names of all *DomainParticipants* within the DDS domain and the exact QoS of the remote *DataWriters* and *DataReaders*.

---

**Note:** *RTI Connext* uses the acronyms SPDP and SEDP to distinguish between the two phases of Simple Discovery: participant and endpoint phases (see Discovery in the Core Libraries User's Manual). *RTI Connext Micro* uses the acronyms DPSE and DPDE to distinguish between the static and dynamic endpoint discovery plugins available in *RTI Connext Micro*. The DPSE plugin implements the SPDP protocol and DPDE implements the SPDP and SEDP protocol.

---

Please refer to the C API Reference and C++ API Reference for the following remote entity assertion APIs:

- DPSE_RemoteParticipant_assert
- DPSE_RemotePublication_assert
- DPSE_RemoteSubscription_assert

### Remote Participant Assertion

Given a local *DomainParticipant*, static discovery requires first the names of remote *DomainParticipants* to be asserted, in order for endpoints on them to match. This is done by calling DPSE_RemoteParticipant_assert with the name of a remote *DomainParticipant*. The name must match the name contained in the participant discovery announcement produced by that *DomainParticipant*. This has to be done reciprocally between two *DomainParticipants* so that they may discover one another.

For example, a *DomainParticipant* has entity name "participant_1", while another *DomainParticipant* has name "participant_2." participant_1 should call DPSE_RemoteParticipant_assert("participant_2") in order to discover participant_2. Similarly, participant_2 must also assert participant_1 for discovery between the two to succeed.

```
/* participant_1 is asserting (remote) participant_2 */
retcode = DPSE_RemoteParticipant_assert(participant_1,
                                        "participant_2");
if (retcode != DDS_RETCODE_OK) {
    printf("participant_1 failed to assert participant_2\n");
```

(continues on next page)

(continued from previous page)

```
    goto done;
}
```

### Remote Publication and Subscription Assertion

Next, a *DomainParticipant* needs to assert the remote endpoints it wants to match that belong to an already asserted remote *DomainParticipant*. The endpoint assertion function is used, specifying an argument which contains all the QoS and configuration of the remote endpoint. Where DPDE gets remote endpoint QoS information from received endpoint-discovery messages, in DPSE, the remote endpoint's QoS must be configured locally. With remote endpoints asserted, the *DomainParticipant* then waits until it receives a participant discovery announcement from an asserted remote *DomainParticipant*. Once received that, all endpoints that have been asserted for that remote *DomainParticipant* are considered discovered and ready to be matched with local endpoints.

Assume participant_1 contains a *DataWriter*, and participant_2 has a *DataReader*, both communicating on topic HelloWorld. participant_1 needs to assert the *DataReader* in participant_2 as a remote subscription. The remote subscription data passed to the operation must match exactly the QoS actually used by the remote *DataReader*:

```
/* Set participant_2's reader's QoS in remote subscription data  */
rem_subscription_data.key.value[DDS_BUILTIN_TOPIC_KEY_OBJECT_ID] = 200;
rem_subscription_data.topic_name = DDS_String_dup("Example HelloWorld");
rem_subscription_data.type_name = DDS_String_dup("HelloWorld");
rem_subscription_data.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

/* Assert reader as a remote subscription belonging to (remote) participant_2 */
retcode = DPSE_RemoteSubscription_assert(participant_1,
                                         "participant_2",
                                         &rem_subscription_data,
                                         HelloWorld_get_key_kind(HelloWorldTypePlugin_
↪get(), NULL));
if (retcode != DDS_RETCODE_OK)
{
    printf("failed to assert remote subscription\n");
    goto done;
}
```

Reciprocally, participant_2 must assert participant_1's *DataWriter* as a remote publication, also specifying matching QoS parameters:

```
/* Set participant_1's writer's QoS in remote publication data  */
rem_publication_data.key.value[DDS_BUILTIN_TOPIC_KEY_OBJECT_ID] = 100;
rem_publication_data.key.value.topic_name = DDS_String_dup("Example HelloWorld");
rem_publication_data.key.value.type_name = DDS_String_dup("HelloWorld");
rem_publication_data.key.value.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

/* Assert writer as a remote publication belonging to (remote) participant_1 */
retcode = DPSE_RemotePublication_assert(participant_2,
```

(continues on next page)

(continued from previous page)

```
                                        "participant_1",
                                        &rem_publication_data,
                                        HelloWorld_get_key_kind(HelloWorldTypePlugin_
↪get(), NULL));
if (retcode != DDS_RETCODE_OK)
{
    printf("failed to assert remote publication\n");
    goto done;
}
```

When participant_1 receives a participant discovery message from participant_2, it is aware of participant_2, based on its previous assertion, and it knows participant_2 has a matching *DataReader*, also based on the previous assertion of the remote endpoint. It therefore establishes a match between its *DataWriter* and participant_2's *DataReader*. Likewise, participant_2 will match participant_1's *DataWriter* with its local *DataReader*, upon receiving one of participant_1's participant discovery messages.

Note, with DPSE, there is no runtime check of QoS consistency between *DataWriters* and *DataReaders*, because no endpoint discovery messages are exchanged. This makes it extremely important that users of DPSE ensure that the QoS set for a local *DataWriter* and *DataReader* is the same QoS being used by another *DomainParticipant* to assert it as a remote *DataWriter* or *DataReader*.

### 3.6.8 Asymmetric Matching and Lost Samples

The DDS discovery process is necessary to establish communication between a *DataWriter* and a *DataReader*. However, it is important to understand that DDS applications do not connect to each other; there is no handshake protocol to ensure that a *DataReader* is ready to receive data from a *DataWriter*. Thus, it is possible that a *DataWriter* matches a *DataReader* before the *DataReader* matches the *DataWriter* (and vice versa). For this reason, it is possible that data published by a *DataWriter* is not received by the *DataReader*, even on a local network.

The reason for this asymmetric behavior can be for any number of reasons, such as, but not limited to:

- Network delays
- Packets taking different paths through the network
- Address resolution delays
- OS scheduling

DDS offers some solutions to mitigate this problem, e.g., the DURABILITY QoS policy, but in other cases it may be necessary for applications to implement their own synchronization protocols.

### 3.6.9 DomainParticipant Discovery by Name

If a *DomainParticipant* is restarted after an ungraceful shutown, other *DomainParticipants* will not remove the *DomainParticipant*'s discovery information until its `lease_duration` has expired. Typically, you can reduce the `lease_duration` or increase the resource limits to avoid running out of resources to discover restarted *DomainParticipants*. However, this may not be a feasible solution, since it increases network load and memory usage.

To mitigate these issues, *Connext Micro* can be configured to discover *DomainParticipants* by name instead of by GUID. This speeds up the discovery process because the restarted *DomainParticipant* can immediately replace the old *DomainParticipant* without waiting for the old one's `lease_duration` to expire.

To enable discovery by name for a *DomainParticipant*, do the following:

1. Set a globally unique `participant_name` in the *DomainParticipant*'s DomainParticipantQos.
2. Set the `enable_participant_discovery_by_name` property to `DDS_BOOLEAN_TRUE` in the *DomainParticipant*'s DISCOVERY QosPolicy.

The *DomainParticipant* can now be rediscovered smoothly by other remote *DomainParticipants* after a restart.

**Attention:** When using discovery by name, a *DomainParticipant* must be restarted with a different GUID from its previous GUID.

**Attention:** To avoid undefined behavior while using discovery by name, make sure that each *DomainParticipant* in a given domain has a unique `participant_name`.

### 3.6.10 Queueing Discovery Messages

During a system restart or partial restart, the discovery process also restarts as *DomainParticipants* come back up. During this period, it is possible that each *DomainParticipant* is in a different phase. For example:

- Some *DomainParticipants* may have terminated gracefully, but have not started up again.
- Some *DomainParticipants* may have been abruptly taken off-line; e.g., during a power cycle.
- Some *DomainParticipants* may have already restarted.
- Some *DomainParticipants* may not have restarted yet, or will not restart.

During this period it is possible for a *DomainParticipant* to temporarily exceed its resource limits. For example, a *DomainParticipant* that was not restarted may store discovery information for a *DomainParticipant* that was ungracefully shut down because its lease duration has not expired, and **also** discover the restarted *DomainParticipant*.

*Connext Micro* will normally discard a discovery message if it receives an endpoint discovery message and lacks sufficient resources to store the endpoint. *Connext Micro* will (by default) acknowledge the message as received, but discard the discovery information. This can lead to a mismatch in discovery states between *DomainParticipants*. However, as lease durations expire on restarted *DomainParticipants*, resources become available to discover new *DomainParticipants*.

To mitigate this temporary lack of resources, you can set the `DomainParticipantQos.discovery.enable_endpoint_discovery_queue` property can be set to `DDS_BOOLEAN_TRUE`. Endpoint discovery messages will then instead be queued for later processing when resources become available.

**Attention:** This new field assumes that there are sufficient resources available for discovery information and that the lack of resources is temporary (such as during a system restart). Setting this value to `DDS_BOOLEAN_TRUE` without sufficient resources may cause undefined behavior.

### 3.6.11 Restarting Discovery

DDS discovery can be configured so that no action is required from the user application once initial discovery is complete; this is the typical use case. However, there may be cases where the user would like to restart discovery due to some runtime event. *Connext Micro* includes two functions to restart *DomainParticipant* discovery:

1. DDS_DomainParticipant_remove_discovered_participants()
2. DDS_DomainParticipant_announce()

To illustrate when a user might benefit from such functionality, this document will explore a system that suspends and resumes CPUs as an example. Because this is a known use case for this feature, we refer to these functions as "suspend-resume" APIs.

For more information regarding the use of these APIs, go to *How to restart discovery*.

#### How CPU suspension affects DDS discovery

Some systems benefit from suspending CPUs when they enter a dormant state, usually to save power. However, applications running on the suspended CPU may be maintaining a state that is time-dependent, and also may not be aware that a suspend or resume operation occurred. This can create a divergence in the system view between applications on the recently-resumed CPU and applications on other CPUs that have been running without pause.

This situation can impact DDS applications because the mechanisms implementing DDS discovery are time-dependent. If a local *DomainParticipant* does not announce itself in a timely manner, remote *DomainParticipants* will assume that the local *DomainParticipant* is no longer present (or "alive") in the domain.

The DDS standard specifies functionality to recover from lapses in communication; namely, the `participant_liveliness_lease_duration` and `participant_liveliness_assert_period` QoS policies. These can be configured to limit the amount of time before a *DomainParticipant* can be rediscovered after being registered as "dead" by remote peers.

However, you may want to tune these parameters for steady-state use cases and still have a way to bring the system back to a known state after suspending and resuming a CPU.

When a CPU is suspended and then later resumed, any DDS applications located on that CPU could face two discovery-related issues:

1. The discovery database of the local *DomainParticipant* may be out-of-sync with the actual reality of remote *DomainParticipants* in the domain.

   While the CPU was suspended, some remote *DomainParticipants* may have shut down and others may have started up. But the newly-resumed local *DomainParticipant* still has a record of the remote *DomainParticipants* that were present at the time of CPU suspension. This local *DomainParticipant* may then waste time and network bandwidth by trying to communicate with remote peers that are no longer present.

2. If the CPU was suspended for longer than the `participant_liveliness_lease_duration` of a local *DomainParticipant* on that CPU, the *DomainParticipant* in question would not have been able to announce itself to any remote *DomainParticipants*. In this case, from a remote peer's perspective, the local *DomainParticipant* has disappeared from the domain.

   Once resumed, any local DDS application may continue publishing or subscribing to topics, but it would be unaware that all remote endpoints have unmatched due to the loss of *DomainParticipant* liveliness.

#### How to restart discovery

To restart discovery from the perspective of the local *DomainParticipant*, call the suspend-resume API functions in order:

1. DDS_DomainParticipant_remove_discovered_participants()

   Calling this function on the local *DomainParticipant* removes all currently-discovered remote *DomainParticipants* from the discovery database. If a removed remote *DomainParticipant* contained endpoints matched with local endpoints, the local endpoints are unmatched from the remote *DomainParticipant* endpoints.

2. DDS_DomainParticipant_announce()

   Calling this function on the local *DomainParticipant* causes a *DomainParticipant* announcement (`DATA(p)`) to be sent. The `DATA(p)` is sent to each peer in the `initial_peers` list and to all currently-discovered *DomainParticipants*.

These two functions can be used to restart the discovery process as shown below:

The APIs are typically used by calling them in the order described above, but they are independent; either API can be used without the other. If you call DDS_DomainParticipant_remove_discovered_participants() only, then the discovery database will be reset and participant announcements will be sent according to the the local participants QoS configuration. If instead DDS_DomainParticipant_announce() is called alone, a participant announcement will be sent immediately to the currently known peers; this is shown in the diagram below:

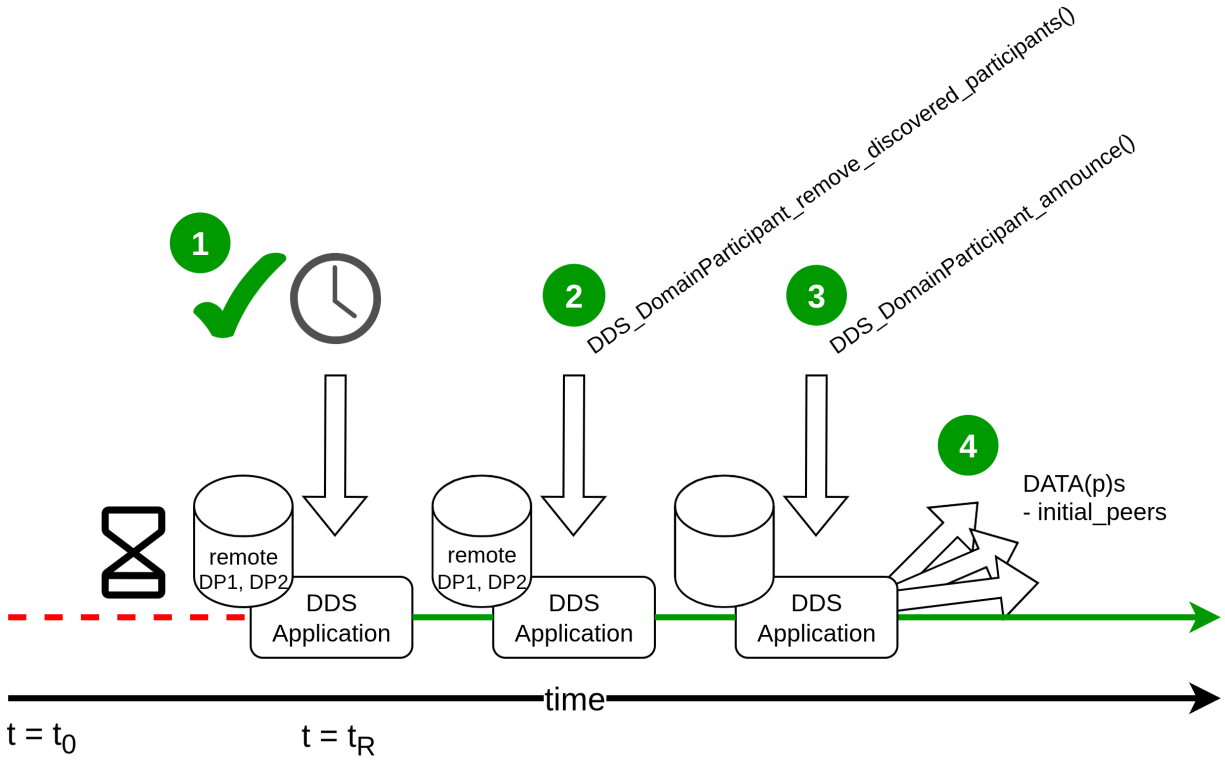



Figure 3.4: Post-resume API usage

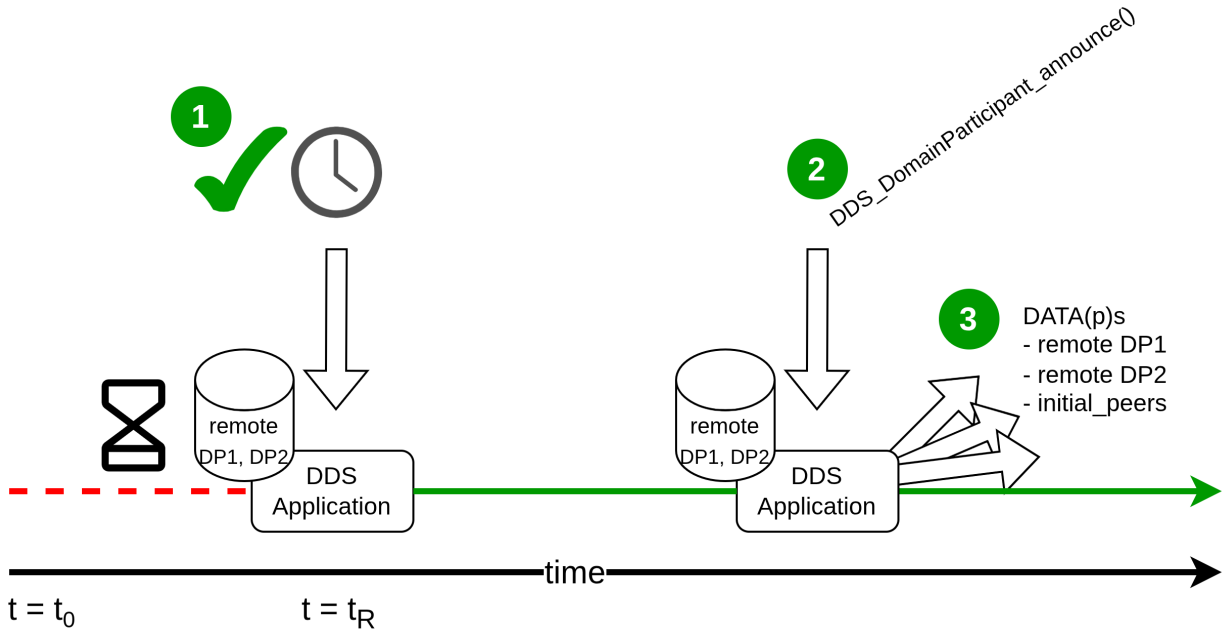



Figure 3.5: Post-resume announcement without database purge

#### Usage considerations

It is up to your application to determine how and when to use these APIs.

In the example above, when the CPU in question resumes, you need to have implemented a mechanism to inform a DDS application that it has recently been through a suspend-resume cycle. This could be a message sent over a UDP socket, a POSIX signal (such as SIGUSR1) that is sent and handled, or some hardware interrupt. At that point, the DDS application can call the suspend-resume APIs to help return the system to its pre-suspension discovery state.

### 3.6.12 Interoperating with Connext Professional Discovery

When trying to establish communication between an *Connext Micro* application that uses the Dynamic Participant Static Endpoint (DPSE) discovery module and an RTI product based on *Connext Professional*, every participant in the DDS system must be configured with a unique participant name. While the static discovery functionality provided by *Connext Professional* allows participants on different hosts to share the same name, *Connext Micro* requires every participant to have a different name to help keep the complexity of its implementation suitable for smaller targets.

Also, *Connext DataWriters* that are configured to send compressed data will not match with *Connext Micro DataReaders*, since *Connext Micro* does not support sending or receiving compressed data. See DATA_REPRESENTATION QosPolicy in the Core Libraries User's Manual for more information on the *Connext* compression feature.

## 3.7 User Discovery Data

### 3.7.1 Introduction

User Discovery Data is a feature of *Connext Micro* that provides areas where your application can store additional information related to DDS *Entities*. How this information is used will be up to user code; *Connext Micro* distributes this information to other applications as part of the discovery process, but *Connext Micro* does not interpret the information. Use cases are usually application-to-application identification, authentication, authorization, and encryption.

There are three User Discovery Data QoS policies:

- USER_DATA: associated with *DomainParticipants*, *DataWriters*, and *DataReaders*.
- TOPIC_DATA: associated with a *Topic*.
- GROUP_DATA: associated with a *Publisher* or *Subscriber*.

**Warning:** These QoS policies must be specified when an entity is created and cannot be modified at runtime.

### 3.7.2 Resource Limits

Before these QoS policies can be used, the DomainParticipantResourceLimitsQosPolicy must be configured to set the maximum length of each kind of data which will be used. These settings are listed below:

- `participant_user_data_max_length`
- `topic_data_max_length`
- `publisher_group_data_max_length`
- `subscriber_group_data_max_length`
- `writer_user_data_max_length`
- `reader_user_data_max_length`

These policy settings limit the length of the data field, and must be configured to the same values for all *DomainParticipants* in the same DDS domain. Attempting to create an entity with user discovery data larger than the corresponding QoS setting will cause creation of that entity to fail. Similarly, discovering remote entities will fail if those entities have user discovery data larger than the QoS policy setting.

Memory usage by the discovery data will be directly affected by the maximum length of each type of data because *Connext Micro* will pre-allocate all of the memory necessary to store received data. Setting the maximum length appropriately will limit the memory usage of this feature.

*Connext Micro* also adds settings to further optimize memory usage via the max_count resource limit options, listed below:

- `participant_user_data_max_count`
- `topic_data_max_count`
- `publisher_group_data_max_count`
- `subscriber_group_data_max_count`
- `writer_user_data_max_count`
- `reader_user_data_max_count`

These options limit the number of unique data of a given type. Data from local and discovered entities both contribute toward this limit.

These maximums will default to `DDS_SIZE_AUTO`, in which case *Connext Micro* will generate an appropriate value based on other QoSes to ensure that discovery will always succeed. However, if you know the maximum number of unique data that will exist in the domain, setting these options accordingly can reduce the total amount of memory allocated.

**Warning:** Once the max_count limit of unique data has been reached for a given entity type, discovery of entities with additional unique data will fail.

### 3.7.3 Propagating User Discovery Data

When using *Dynamic Discovery Plugin* (DPDE), the information associated with all entities is automatically passed between applications during discovery using builtin topics. When using *Static Discovery Plugin* (DPSE), only the USER_DATA associated with a participant will be automatically passed between applications; for remote *DataReaders* and *DataWriters*, the associated USER_DATA, TOPIC_DATA, or GROUP_DATA must be asserted with remote publications and subscriptions. (See *Accessing User Discovery Data* below.)

For example, to assert USER_DATA associated with a remote subscription in DPSE:

```
struct DDS_SubscriptionBuiltinTopicData rem_subscription_data =
   DDS_SubscriptionBuiltinTopicData_INITIALIZER;

/* Set Reader's protocol.rtps_object_id */
rem_subscription_data.key.value[DDS_BUILTIN_TOPIC_KEY_OBJECT_ID] = 200;

rem_subscription_data.topic_name = DDS_String_dup("Example HelloWorld");
rem_subscription_data.type_name = DDS_String_dup("HelloWorld");

rem_subscription_data.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

/* Set USER_DATA */
DDS_OctetSeq_set_maximum(&rem_subscription_data.user_data.value, 2);
DDS_OctetSeq_set_length(&rem_subscription_data.user_data.value, 2);
*DDS_OctetSeq_get_reference(&rem_subscription_data.user_data.value, 0) = 0xAA;
*DDS_OctetSeq_get_reference(&rem_subscription_data.user_data.value, 1) = 0xBB;

retcode = DPSE_RemoteSubscription_assert(participant,
                                         "Participant_2",
                                         &rem_subscription_data,
                                         HelloWorld_get_key_kind(HelloWorldTypePlugin_
↪get(),
                                         NULL));
if (retcode != DDS_RETCODE_OK)
{
   /* failure */
}
```

### 3.7.4 Accessing User Discovery Data

Whether using DPDE or DPSE, the USER_DATA, TOPIC_DATA, and GROUP_DATA is propagated with the information about remote *DomainParticipants*, *DataWriters*, and *DataReaders*. For *DomainParticipants*, the associated USER_DATA can be accessed through ParticipantBuiltinTopicData. For *DataWriters* and *DataReaders*, the associated USER_DATA, TOPIC_DATA, and GROUP_DATA can be accessed through the PublicationBuiltinTopicData and SubscriptionBuiltinTopicData respectively.

For *DomainParticipants*, the discovery information for discovered participants can be accessed through the get_discovered APIs:

- DDS_DomainParticipant_get_discovered_participants
- DDS_DomainParticipant_get_discovered_participant_data

For *DataReaders* and *DataWriters*, the information on matched entities can be retrieved through the get_matched APIs:

- DDS_DataWriter_get_matched_subscriptions
- DDS_DataWriter_get_matched_subscription_data
- DDS_DataReader_get_matched_publications
- DDS_DataReader_get_matched_publication_data

Note that these APIs will perform a copy into the provided data sample. If the provided sample does not have enough memory to store the data, additional memory will be allocated to fit it. This memory can instead be pre-allocated with the corresponding initialize_from_qos function in C. If a sample is pre-allocated based on the configured QoS, then no additional memory will need to be allocated to perform the copy.

The C++ API does not support initialize_from_qos. The default constructors for the builtin topic data types will not allocate any memory for the user discovery data sequences. The overloaded constructor accepts a *DomainParticipantQos* and initializes the memory according to the resource limits, including memory for DDS_ParticipantBuiltinTopicData(), DDS_PublicationBuiltinTopicData(), and DDS_SubscriptionBuiltinTopicData().

For example, to retrieve information on discovered participants:

```
struct DDS_InstanceHandleSeq handles = DDS_SEQUENCE_INITIALIZER;
struct DDS_InstanceHandle handle;
struct DDS_DomainParticipantQos dp_qos =
            DDS_DomainParticipantQos_INITIALIZER;
struct DDS_ParticipantBuiltinTopicData dp_data =
            DDS_ParticipantBuiltinTopicData_INITIALIZER;

/* Pre-allocate memory for discovery data based on participant's QoS */
retcode = DDS_DomainParticipant_get_qos(participant, &dp_qos);
if (retcode != DDS_RETCODE_OK)
{
   /* failure */
}
if (!DDS_ParticipantBuiltinTopicData_initialize_from_qos(&dp_data, &dp_qos))
{
   /* failure */
}

/* Get instance handles of discovered participants */
retcode = DDS_DomainParticipant_get_discovered_participants(participant, &handles);
if (retcode != DDS_RETCODE_OK)
{
   /* failure */
}
```

(continues on next page)

(continued from previous page)

```
/* For each handle, get the discovery data */
for (DDS_Long j = 0; j < DDS_InstanceHandleSeq_get_length(&handles); ++j)
{
   handle = DDS_InstanceHandleSeq_get_reference(&handles, j);
   if (handle == NULL)
   {
      /* failure */
   }

   retcode = DDS_DomainParticipant_get_discovered_participant_data(participant,
                                                                   &dp_data,
                                                                   handle);
   if (retcode != DDS_RETCODE_OK)
   {
      /* failure */
   }
   else
   {
      /* Discovered participant USER_DATA can be accessed in dp_data.user_data */
   }
}
```

### 3.7.5 QoS Policies

#### USER_DATA

This Qos Policy provides an area where your application can store additional information related to a *DomainParticipant*, *DataWriter*, or *DataReader*.

You will need to access the value of USER_DATA through ParticipantBuiltinTopicData, PublicationBuiltinTopicData or SubscriptionBuiltinTopicData. (See *Accessing User Discovery Data*.)

The structure for the USER_DATA QosPolicy includes just one field, as seen in Table 3.1. The field is a sequence of octets that translates to a contiguous buffer of bytes whose contents and length are set by the user. The maximum size for the data is set in the DomainParticipantResourceLimitsQosPolicy. (See *Resource Limits*.)

Table 3.1: DDS_UserDataQosPolicy

| Type | Field Name | Description |
|---|---|---|
| DDS_OctetSeq | value | Empty by default |

#### TOPIC_DATA

This QoS Policy provides an area where your application can store additional information related to the *Topic*.

Currently, TOPIC_DATA of the associated *Topic* is only propagated with the information that declares a *DataWriter* or *DataReader*. Thus, you will need to access the value of TOPIC_DATA through PublicationBuiltinTopicData or SubscriptionBuiltinTopicData. (See *Accessing User Discovery Data*)

The structure for the TOPIC_DATA QosPolicy includes just one field, as seen in Table 3.2. The field is a sequence of octets that translates to a contiguous buffer of bytes whose contents and length are set by the user. The maximum size for the data is set in the DomainParticipantResourceLimitsQosPolicy. (See *Resource Limits*)

Table 3.2: DDS_TopicDataQosPolicy

| Type | Field Name | Description |
|---|---|---|
| DDS_OctetSeq | value | Empty by default |

#### GROUP_DATA

This Qos Policy provides an area where your application can store additional information related to the *Publisher* and *Subscriber*.

Currently, GROUP_DATA of the associated *Publisher* or *Subscriber* is only propagated with the information that declares a *DataWriter* or *DataReader*. Thus, you will need to access the value of GROUP_DATA through PublicationBuiltinTopicData or SubscriptionBuiltinTopicData. (See *Accessing User Discovery Data*)

The structure for the TOPIC_DATA QosPolicy includes just one field, as seen in Table 3.3. The field is a sequence of octets that translates to a contiguous buffer of bytes whose contents and length are set by the user. The maximum size for the data is set in the DomainParticipantResourceLimitsQosPolicy. (See *Resource Limits*)

Table 3.3: DDS_GroupDataQosPolicy

| Type | Field Name | Description |
|---|---|---|
| DDS_OctetSeq | value | Empty by default |

## 3.8 Partitions

### 3.8.1 Introduction

The PARTITION QoS policy provides a way to control which *Entities* will match—and thus communicate with—which other *Entities*. It can be used to prevent *Entities* that would have otherwise matched from talking to each other. Much in the same way that only applications within the same

DDS domain will communicate with each other, only *Entities* that belong to the same partition can talk to each other.

The PARTITION QoS policy applies to *Publishers* and *Subscribers*. *DataWriters* and *DataReaders* belong to the partitions as set in the QoS of the *Publishers* and *Subscribers* that created them.

The PARTITION QoS policy consists of a set of partition names that identify the partitions of which the *Entity* is a member. These names can be concrete (e.g., ExamplePartition) or regular expression strings (e.g, Example*), and two *Entities* are considered to be in the same partition if one of the *Entities* has a concrete partition name matching one of the concrete or regular expression partition names of the other *Entity* (see *Pattern matching for partition names*). By default, *DataWriters* and *DataReaders* (through their *Publisher/Subscriber* parents), belong to a single partition whose name is the empty string, "".

Conceptually, each partition name can be thought of as defining a "visibility plane" within the DDS domain. *DataWriters* will make their data available on all of the visibility planes that correspond to their *Publisher's* partition names, and the *DataReaders* will see the data that is placed on all of the visibility planes that correspond to their *Subscriber's* partition names.

Figure 3.6 illustrates the concept of a PARTITION QoS policy at the *Publisher* and *Subscriber* level. In this figure, all *DataWriters* and *DataReaders* belong to the same DDS domain ID and *DomainParticipant* partition, and they use the same *Topic*. *DataWriter1* is configured to belong to three partitions: partition_A, partition_B, and partition_C. *DataWriter2* belongs to partition_C and partition_D.

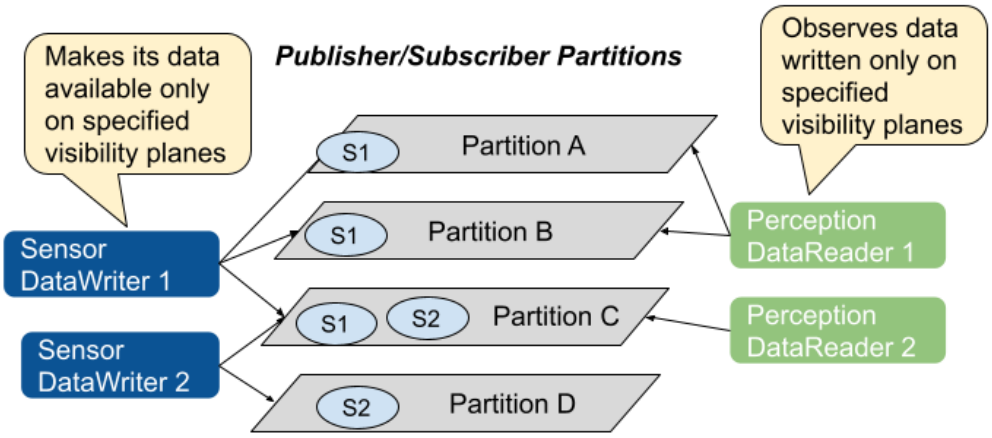



Figure 3.6: Controlling Visibility of Data with PARTITION QoS

Similarly, *DataReader1* is configured to belong to partition_A and partition_B, and *DataReader2* belongs only to partition_C. Given this topology, the data written by *DataWriter1* is visible in partitions A, B, and C. The oval tagged with the number "S1" represents one DDS data sample written by *DataWriter1*.

Similarly, the data written by *DataWriter2* is visible in partitions C and D. The oval tagged with the number "S2" represents one DDS data sample written by *DataWriter2*.

The result is that the data written by *DataWriter1* will be received by both *DataReader1* and *DataReader2*, but the data written by *DataWriter2* will only be visible by *DataReader2*.

*Publishers* and *Subscribers* always belong to a partition. By default, *Publishers* and *Subscribers* belong to a single partition whose name is the empty string, "". If you set the PARTITION QoS policy to be an empty set, *Connext Micro* will assign the *Publisher* or *Subscriber* to the default partition, "". Thus, for the example above, without using the PARTITION QoS policy on any of the entities, *DataReaders* 1 and 2 would have received all data samples written by *DataWriters* 1 and 2.

### 3.8.2 Rules for partition matching

The PARTITION QoS Policy associates a set of partition names with the entity (*Publisher* or *Subscriber*). The partition names are concrete names (e.g., ExamplePartition) or regular expression strings (e.g, Example*).

With regard to the PARTITION QoS Policy, a *DataWriter* will communicate with a *DataReader* if and only if the following conditions apply:

1. The *DataWriter* and *DataReader* belong to *DomainParticipants* bound to the same DDS domain ID.
2. The *DataWriter* and *DataReader* have matching *Topics*. That is, each is associated with a *Topic* with the same name and compatible data type.
3. The QoS policy offered by the *DataWriter* is compatible with the QoS requested by the *DataReader*.
4. The *DataWriter* and *DataReader* have a matching partition name.

Matching partition names is done by string pattern matching, and partition names are case-sensitive.

---

**Note:** Failure to match partitions (on *Publisher* or *Subscriber*) is not considered an incompatible QoS policy and does not trigger any listeners or change any status conditions.

---

### 3.8.3 Pattern matching for partition names

You may add strings that are regular expressions to the PARTITION QosPolicy. A `PARTITION.name` is a regular expression if it contains any of the following unescaped special characters: `*`, `?`, `[`, `]`, `!`, or `^`. The `PARTITION.name` strings can be "concrete" names or regular expression strings; a `PARTITION.name` element that is a regular expression will only match against concrete strings found in a `PARTITION.name` element of a different *Entity's* PARTITION QosPolicy.

If a PARTITION QoS policy only contains regular expressions, then the *Entity* will be assigned automatically to the default partition with the empty string name (""). Thus, a PARTITION QoS policy that only contains the string `*` matches another *Entity's* PARTITION QoS policy that also only contains the string `*`, not because the regular expression strings are identical, but because they both belong to the default "" partition.

For more on regular expressions, see *Regular Expression Matching* below.

Two *Entities* are considered to have a partition in common if the sets of partitions associated with them have:

- At least one concrete partition name in common
- A regular expression in one *Entity* that matches a concrete partition name in another *Entity*

The programmatic representation of the PARTITION QoS policy is shown in Table 3.4. The Qos policy contains the single string sequence, name. Each element in the sequence can be a concrete name or a regular expression. The *Entity* will be assigned to the default "" partition if the sequence is empty, or if the sequence contains only regular expressions.

Table 3.4: DDS_PartitionQosPolicy

| Type | Field Name | Description |
|---|---|---|
| DDS_StringSeq | name | Empty by default.<br>There can be up to 64 names, with a maximum of 256 characters (including the NUL terminator), summed across all names. |

You can have one long partition string of 256 chars, or multiple shorter strings that add up to 256 or fewer characters. For example, you can have one string of 4 chars and one string of 252 chars.

### Regular Expression Matching

The SQL expression format provided by *Connext Micro* supports the relational operator **MATCH**. It may only be used with string fields. The right-hand operator is a string pattern, which specifies a template that the left-hand field must match.

**MATCH** is case-sensitive. The following characters have special meaning, unless escaped by the escape character: `,\/?*[]-^!\%`.

The pattern allows limited "wild card" matching under the rules in Table 3.5.

The syntax is similar to the POSIX® fnmatch syntax. (See http://www.opengroup.org/onlinepubs/000095399/functions/fnmatch.html.)

Table 3.5: Wild Card Matching

| Character | Meaning |
|---|---|
| , | **NOT SUPPORTED**<br>A , separates a list of alternate patterns. The field string is matched if it matches one or more of the patterns. |
| / | **NOT SUPPORTED**<br>A / in the pattern string matches a / in the field string. It separates a sequence of mandatory substrings. |
| ? | A ? in the pattern string matches any single non-special characters in the field string. |
| * | A * in the pattern string matches 0 or more non-special characters in the field string. |
| % | **NOT SUPPORTED**<br>This special character is used to designate filter expression parameters. |
| \ | Escape character for special characters. |
| [charlist] | Matches any one of the characters in charlist. |
| [!charlist] or [^charlist] | Matches any one of the characters *not* in charlist. |
| [s-e] | Matches any character from **s** to **e**, inclusive. |
| [!s-e] or [^s-e] | Matches any character *not* in the interval **s** to **e**. |

**Note:** To use special characters as regular characters in regular expressions, you must escape them using the character \. For example, `A[` is considered a malformed expression and the result is undefined.

### 3.8.4 Example

The PARTITION Qos policy is useful to control which *DataWriters* can communicate with which *DataReaders* and vice versa—even if all of the *DataWriters* and *DataReaders* are for the same *Topic*. This facility is useful for creating temporary separation groups among *Entities* that would otherwise be connected to and exchange data each other.

The code below illustrates how to set the PARTITION Qos policy on a *Publisher*:

```
struct DDS_PublisherQos pub_qos = DDS_PublisherQos_INITIALIZER;
   DDS_StringSeq_set_maximum(&pub_qos.partition.name,2);
   DDS_StringSeq_set_length(&pub_qos.partition.name,2);
   *DDS_StringSeq_get_reference(&pub_qos.partition.name,0) = DDS_String_dup("partition1
↪");
   *DDS_StringSeq_get_reference(&pub_qos.partition.name,1) = DDS_String_dup("partition2
↪");

publisher = DDS_DomainParticipant_create_publisher(application->participant,&pub_qos,␣
↪NULL,DDS_STATUS_MASK_NONE);
   if (publisher == NULL)
```

(continues on next page)

(continued from previous page)

```
{
   ...
}
```

Using partitions, connectivity can be controlled based on location-based partitioning, access-control groups, or a combination of these and other application-defined criteria. We will examine some of these options via concrete examples.

### Location-based partitions

Assume you have a set of *Topics* in a traffic management system such as "TrafficAlert," "AccidentReport," and "CongestionStatus." You may want to control the visibility of these *Topics* based on the actual location to which the information applies. You can do this by placing the *Publisher* in a partition that represents the area to which the information applies. This can be done using a string that includes the city, state, and country, such as "USA/California/Santa Clara." A *Subscriber* can then choose whether it wants to see the alerts in a single city, the accidents in a set of states, or the congestion status across the US. Some concrete examples are shown in Table 3.6.

Table 3.6: Example of Using Location-Based Partitions

| Publisher Partitions | Subscriber Partitions | Result |
|---|---|---|
| Specify a single partition name using the pattern: "<country>/<state>/<city>" | Specify multiple partition names, one per region of interest | Limits the visibility of the data to *Subscribers* that express interest in the geographical region. |
| "USA/California/Santa Clara" | (*Subscriber* partition is irrelevant here.) | Send only information for Santa Clara, California. |
| (Publisher partition is irrelevant here.) | "USA/California/Santa Clara" | Receive only information for Santa Clara, California. |
| (Publisher partition is irrelevant here.) | "USA/California/Santa Clara"<br>"USA/California/Sunnyvale" | Receive information for Santa Clara or Sunnyvale, California. |
| (Publisher partition is irrelevant here.) | "USA/California/*"<br>"USA/Nevada/*" | Receive information for California or Nevada. |
| (Publisher partition is irrelevant here.) | "USA/California/*"<br>"USA/Nevada/Reno"<br>"USA/Nevada/Las Vegas" | Receive information for California and two cities in Nevada. |

#### Access-control group partitions

Suppose you have an application where access to the information must be restricted based on reader membership to access-control groups. You can map this group-controlled visibility to partitions by naming all the groups (e.g., executives, payroll, financial, general-staff, consultants, external-people) and assigning the *Publisher* to the set of partitions that represents which groups should have access to the information. The *Subscribers* specify the groups to which they belong, and the partition-matching behavior will ensure that the information is only distributed to *Subscribers* belonging to the appropriate groups. Some concrete examples are shown in Table 3.7

Table 3.7: Example of Access-Control Group Partitions

| Publisher Partitions | Subscriber Partitions | Result |
|---|---|---|
| Specify several partition names, one per group that is allowed access: | Specify multiple partition names, one per group to which the *Subscriber* belongs. | Limits the visibility of the data to *Subscribers* that belong to the access-groups specified by the *Publisher*. |
| "payroll"<br>"financial" | (*Subscriber* partition is irrelevant here.) | Makes information available only to *Subscribers* that have access to either financial or payroll information. |
| (*Publisher* partition is irrelevant here.) | "executives"<br>"financial" | Gain access to information that is intended for executives or people with access to the finances. |

A slight variation of this pattern could be used to confine the information based on security levels.

### 3.8.5 Properties

This Qos policy cannot be modified at runtime.

Strictly speaking, this Qos policy does not have request-offered semantics, although it is matched between *DataWriters* and *DataReaders*, and communication is established only if there is a match between partition names.

### 3.8.6 Resource limits

Before this QoS policy can be used, you must configure the following DomainParticipantResourceLimitsQosPolicy fields:

- `max_partitions`: sets the maximum number of partitions for each PARTITION QoS policy.
- `max_partition_cumulative_characters`: sets the maximum number of characters (per *DomainParticipant*) that can be used for the sum-total length of all partition names. Note that the NUL terminator in each string contributes to the character count.
- `max_partition_string_size`: sets the maximum number of characters that can be used for each partition name. This can be set to a value greater than 0 or DDS_LENGTH_UNLIMITED.

- `max_partition_string_allocation`: sets the maximum total memory allocated to partition names across all *DomainParticipants*. This can be set to a value greater than 0 or DDS_LENGTH_UNLIMITED.

---

**Note:** All applications in the DDS domain must have the same resource limit values in order to communicate. For example, if two applications have different values, and one application sets the PARTITION Qos policy to hold more partitions or longer names than set by another application, the matching *Entities* between the two applications will not connect. This is similar to the restrictions for the *GROUP_DATA*, *USER_DATA*, and *TOPIC_DATA* Qos Policies.

---

These fields collectively determine how your application manages partition memory. The subsections below explain how to configure your DomainParticipantResourceLimitsQosPolicy for different behaviors.

#### Configuring for runtime allocation

This configuration allows memory for `PARTITION.name` strings to be allocated and freed during runtime. Each `PARTITION.name` string can be of any size; however, the sum-total string length of all partition names is still limited by `max_partition_cumulative_characters`.

For this behavior, set the following:

- Set `max_partition_string_size` to DDS_LENGTH_UNLIMITED.
- Set `max_partition_string_allocation` to DDS_LENGTH_UNLIMITED.

#### Configuring for preallocated memory

Preallocating memory gives you greater control over memory utilization. There are two possible configurations for preallocated memory: reusable and non-reusable.

#### Reusable preallocated memory

In this configuration, memory for `PARTITION.name` strings is preallocated and will never be freed during operation. However, memory allocated to partition names that are internally deleted will be reused for newly-added partition names. For example, if the application creates a *Publisher* with a unique partition name instance and then deletes it, the application will reuse the memory that was storing the unique name (unless there are other uses of that name).

For this behavior, set the following:

- Set `max_partition_string_size` to a value greater than 0.
- Set `max_partition_string_allocation` to a value greater than 0. This value must be large enough to store every instance of each partition name that will be created or discovered.

Note that each partition name will take up memory equal to `max_partition_string_size`, regardless of the actual string length.

#### Non-reusable preallocated memory

In this configuration, memory for `PARTITION.name` strings is preallocated and will never be freed or reused.

For this behavior, set the following:

- Set `max_partition_string_size` to DDS_LENGTH_UNLIMITED.
- Set `max_partition_string_allocation` to a value greater than 0. This value must be large enough to store every instance of each partition name that is created and discovered.

Note that each partition name will take up memory equal to its exact string size.

## 3.9 Content Filtering

Content filtering provides a mechanism for *Connext Micro* to filter out data samples for *DataReaders* based on their contents. The user-specified filtering properties consist of a filter expression and a set of parameters:

- The filter expression evaluates a logical expression on the data samples. The filter expression is similar to the WHERE clause in an SQL expression.
- The parameters are strings that give values to the 'parameters' in the filter expression. There must be one parameter string for each parameter in the filter expression.

Content filtering in *Connext Micro* functions similarly to Filtering Data in *Connext Professional*, except that *Connext Micro* does not use ContentFilteredTopics. Instead, it functions by applying a QoS policy (DDS_ContentFilterQosPolicy) to a *DataReader*. Some limitations apply to *SQL filter expression notation* as well.

### 3.9.1 Creating a content filter

This section will walk you through how to set up content filtering for a *DataReader* in your application.

---

**Note:** When using the C++ API, register the content filtering library using the same method as the C API.

---

1. Register the `DDS_FilterPluginFactory` as shown below:

```
#include "dds_filter/dds_filter.h"

if (DDS_FilterPluginFactory_register() != DDS_RETCODE_OK)
{
    /* Error handling */
}
```

This enables the content filtering feature for your application. It must be registered before a *DomainParticipant* is created in order for it to use the feature.

2. *Optional.* You can configure content filtering support for a given *DomainParticipant*, as shown in the snippet below:

```
struct DDS_DomainParticipantQos dp_qos = DDS_DomainParticipantQos_INITIALIZER;

/* To disable content filtering support for a specific participant,
 * clear the filter plugin name in the DDS_FilterQosPolicy
 */
RT_ComponentFactoryId_clear(&dp_qos.filter.name);

/* If you know that a DomainParticipant will only need to support filtering
 * for a limited number of entities, then you can reduce memory usage
 * by specifying the maximum number of unique filter expressions which
 * can be supported at one time.
 */
dp_qos.filter.resource_limits.filter_expression_max_count = ...;

/* The DDS_FilterQosPolicy has default limits for the maximum length of a filter␣
↪expression,
 * the maximum number of parameters per expression, and the maximum length of a␣
↪filter
 * parameter. Here you can raise these limits if needed to support your use case or␣
↪they
 * can be reduced to limit memory usage.
 */
dp_qos.filter.resource_limits.filter_expression_max_length = ...;
dp_qos.filter.resource_limits.filter_parameter_max_count_per_expression = ...;
dp_qos.filter.resource_limits.filter_parameter_max_length = ...;

/* To reduce memory usage at the cost of network bandwidth, writer-side
 * filtering can also be disabled for all DataWriters created by this participant.
 */
dp_qos.filter.disable_writer_filtering = DDS_BOOLEAN_TRUE;

...  /* Create the DomainParticipant with the given Qos */
```

To customize the maximum number of predicates per SQL expression, disable the built-in SQL filter and register it manually with a `DDS_SqlFilterProperty`:

```
/* Disable the built-in SQL filter */
dp_qos.filter.disable_builtin_sql_filter = DDS_BOOLEAN_TRUE;
/* After creating the participant, register the SQL filter manually: */
struct DDS_SqlFilterProperty sql_prop = DDS_SqlFilterProperty_INITIALIZER;
sql_prop.predicate_max_count_per_expression = 8;
DDS_DomainParticipant_register_contentfilter(
        participant, DDS_SQL_CONTENT_FILTER_CLASS,
        DDS_SqlContentFilter_get_interface(), &sql_prop);
```

The `DDS_DomainParticipantQos` contains a `DDS_FilterQosPolicy` for configuring properties of the content filtering feature for a given *DomainParticipant*. By default, if the

`FilterPluginFactory` is registered, every *DomainParticipant* within the application will support content filtering for all of its child entities. Through the `FilterQosPolicy`, you can change this default behavior to disable support or limit memory usage.

3. Create a *DataReader* with a content filter.

   To enable content filtering on a *DataReader*, you must configure the DDS_ContentFilterQosPolicy in the *DataReader*'s QoS policy. A content filter can either be configured when the *DataReader* is created or it can be added/updated by updating the QoS of the *DataReader*.

   Generally, *Connext Micro* does not support changing the QoS of DDS entities after they are enabled. The `ContentFilterQosPolicy` is an exception in that it can be updated after a *DataReader* is enabled by using `DDS_DataReader_set_qos()`. However, it will fail to update if any other fields of the QoS do not match those specified when the *DataReader* was created.

   The contents of the DDS_ContentFilterQosPolicy are as follows:

| Field | Type | Description |
|---|---|---|
| `filter_name` | DDS_String | A name to uniquely identify this content filter within a *DomainParticipant*. For interoperability with *Connext Professional*, it is equivalent to the `filter_name` when configuring a content filter with XML application creation. |
| `filter_class_name` | DDS_String | The name that identifies the type of filter expression. *Connext Micro* includes a built-in "DDSSQL" filter class. Custom filter classes can be registered using `DDS_DomainParticipant_register_contentfilter()`. |
| `filter_expression` | DDS_String | A logical expression on the contents of the *DataReader*'s *Topic*. If the expression evaluates to TRUE, a DDS sample is received; otherwise, it is discarded. The notation for this expression depends on the filter that you are using (specified by the `filter_class_name` parameter). See *SQL filter expression notation*. |
| `expression_parameters` | DDS_StringSeq | A sequence of filter expression parameters. Each parameter corresponds to a positional argument in the filter expression: element 0 corresponds to positional argument 0, element 1 to positional argument 1, and so forth. |

   For example, to create a *DataReader* with a content filter:

```
struct DDS_DataReaderQos dr_qos = DDS_DataReaderQos_INITIALIZER;

/* Configure the content filter as desired */
dr_qos.content_filter.filter_name = DDS_String_dup("MyFilter");
dr_qos.content_filter.filter_class_name = DDS_String_dup(DDS_SQL_CONTENT_FILTER_
↪CLASS);
dr_qos.content_filter.filter_expression = DDS_String_dup("a < 10");

... /* Create the DataReader with the given Qos */
```

**Warning:** All strings in the DDS_ContentFilterQosPolicy are expected to be owned by the policy. When the DDS_DataReaderQos is finalized, any non-NULL strings in the policy will be freed. The recommended way to set these strings is by allocating space using `DDS_String_dup()`. Otherwise, if you use custom memory management, you must ensure that the strings are set to NULL before finalizing the QoS.

**Note:** Content filtering can also be configured via *Application Generation Using XML*.

If your *DataReader* fails to be created or updated, check if any of the following is true:

- Your *DataReader* specifies a content filter, but content filtering is not enabled on its parent *DomainParticipant*.
- The filter expression exceeds one of the filter resource limits in the DDS_FilterQosPolicy.
- The total number of unique filter expressions within the *DomainParticipant* would exceed the `filter_expression_max_count`.

### 3.9.2 Where filtering is applied

Content filtering can be applied on either side of your distributed application. We refer to filtering applied to a *DataReader* as *reader-side filtering*, while filtering applied for a *DataReader* by a *DataWriter* is called *writer-side filtering*.

Once a *DataReader* configures a filter in its QoS, it will only receive samples that pass the filter. Whether or not that filter is applied by the reader or the writer depends on whether the *DataWriter* supports writer-side filtering. If the writer supports writer-side filtering, then filtered samples can skip being sent over the network. If the *DataWriter* does not support writer-side filtering, then the *DataReader* will continue to only receive samples that pass its filter, but the *DataWriter* will send all samples over the network.

Therefore, writer-side filtering can significantly reduce bandwidth by only sending samples over the network that pass the *DataReader*'s filter. To perform writer-side filtering, the *DataWriter* obtains the filter expression and parameters from the *DataReader* during discovery.

A *DataWriter* will automatically filter DDS data samples for a *DataReader* if all of the following are true:

1. The content filtering feature is enabled on the parent *DomainParticipant*.
2. `dp_qos.filter.disable_writer_filtering` is set to `DDS_BOOLEAN_FALSE` on the parent *DomainParticipant*.
3. The *DataWriter* is filtering for no more than `writer_resource_limits.max_remote_reader_filters` *DataReaders* at the same time.
    a. There is a resource-limit on the *DataWriter* called `writer_resource_limits.max_remote_reader_filters`. This value can be from `[0, (2^31)-1]` or `DDS_LENGTH_UNLIMITED` (default). `0` means do not filter any *DataReader*; `1` to `(2^31)-1`

means that the *DataWriter* will filter for up to the specified number of *DataReaders*; DDS_LENGTH_UNLIMITED means that the *DataWriter* will filter for up to `writer_resource_limits.max_remote_readers`. The *DataWriter* will store the filter result per sample per filtered *DataReader*.

b. If a *DataWriter* is filtering `max_remote_reader_filters` *DataReaders* at the same time and a new filtered *DataReader* is discovered, then the newly created *DataReader* (`max_remote_reader_filters + 1`) will not be filtered by the *DataWriter* (but the *DataReader* will still apply the filter itself).

Even if one of the first (`max_remote_reader_filters`) *DataReaders* is deleted, the already-created *DataReader* (`max_remote_reader_filters + 1`) will still not be filtered. However, any subsequently created *DataReaders* will be filtered as long as the number of *DataReaders* currently being filtered is not more than `writer_resource_limits.max_remote_reader_filters`.

4. There are no more than four matching *DataReaders* in the same locator (transport destination, for example: IP address + port).

---

**Note:** *Connext Micro* supports limited writer-side filtering if there are more than four matching *DataReaders* in the same locator. The middleware will not send any sample to a locator if the sample is filtered out by all the *DataReaders* receiving samples on that locator. However, if there are one or more *DataReaders* to which the sample has to be sent, some *DataReaders* on the locator will perform reader-side filtering for the incoming sample.

---

5. The *DataWriter* has infinite liveliness.
6. The *DataWriter* is not using Asynchronous Publication. Either the *DataWriter*'s `PUBLISH_MODE` kind is set to `DDS_SYNCHRONOUS_PUBLISH_MODE_QOS`, or the kind is set to `DDS_AUTOMATIC_PUBLISH_MODE_QOS` and the data type does not require fragmentation.
7. Support for the specific filter class is registered in the *DomainParticipant* of the *DataWriter*.

   SQL filter support is enabled by default when filtering is enabled on a participant.
8. The specific sample being filtered was written by the *DataWriter* after matching with the *DataReader*.

   If using a `DURABILITY` QoS setting greater than volatile, then samples written before the *DataWriter* matched with the *DataReader* may be available to the *DataReader* but will not have filtering applied by the *DataWriter*.
9. The sample doesn't exceed any of the expression-specific limits or the `filter_expression_max_count` within the *DomainParticipant*.

If any of the above criteria is not met, the *DataReader* will perform reader-side content filtering.

### 3.9.3 Interoperability

Enabling a content filter on a *DataReader* has no impact on the interoperability of that *DataReader* with any other *DataWriter*. However, writer-side filtering is not supported by all other DDS products.

#### With Connext Professional

*Connext Micro*'s implementation of content filtering is compliant with the RTPS specification, allowing other DDS implementations (such as *Connext Professional*) to apply writer-side filtering for a *Connext Micro* application. Similarly, a *Connext Micro* application can apply writer-side filtering for a *Connext Professional* application as long as it meets the criteria listed in *Where filtering is applied*.

*Connext Micro* supports a subset of the SQL filter syntax supported by *Connext Professional*. Any valid SQL filter on a *Connext Micro DataReader* can be applied by a *Connext Professional DataWriter*. However, if a *Connext Professional DataReader* specifies a filter with syntax not supported by *Connext Micro*, then *Connext Micro* will be unable to apply writer-side filtering for that *DataReader*. Refer to *SQL filter expression notation* for more details.

#### With Connext Cert

The built-in SQL content filter is not supported when using the CERT profile. However, custom filters are supported with CERT. See *Custom filters* for how to implement and register a custom filter.

### 3.9.4 SQL filter expression notation

Content filtering uses logical SQL filter expressions, similar to the **WHERE** clause in SQL. The following sections provide more information.

**Attention:** Compared to ContentFilteredTopics in *Connext Professional*, SQL filter expressions in *Connext Micro* have the following limitations:

- The top-level data type must be a struct (unions as the Topic type are not supported).
- You cannot filter on the following members: strings, sequences, arrays, unions, and optional members.
- The XTypes library (`rti_me_ddsxtypes`) is required and a type must be compiled with the interpreter. See *Extensible Types (XTypes) 1.2 Compatibility*.
- Limited syntax; parentheses are not supported.
- The flat data language binding is not supported. Only C and C++ type bindings can be used with the SQL filter.

Future *Connext Micro* releases may remove these limitations.

### Example SQL filter expressions

Assume that you have a *Topic* with two floats, X and Y, which are the coordinates of an object moving inside a rectangle measuring 200 x 200 units. This object moves quite a bit, generating lots of DDS samples that you are not interested in. Instead you only want to receive DDS samples outside the middle of the rectangle, as seen in Figure 3.7. That is, you want to filter out data points in the gray box.

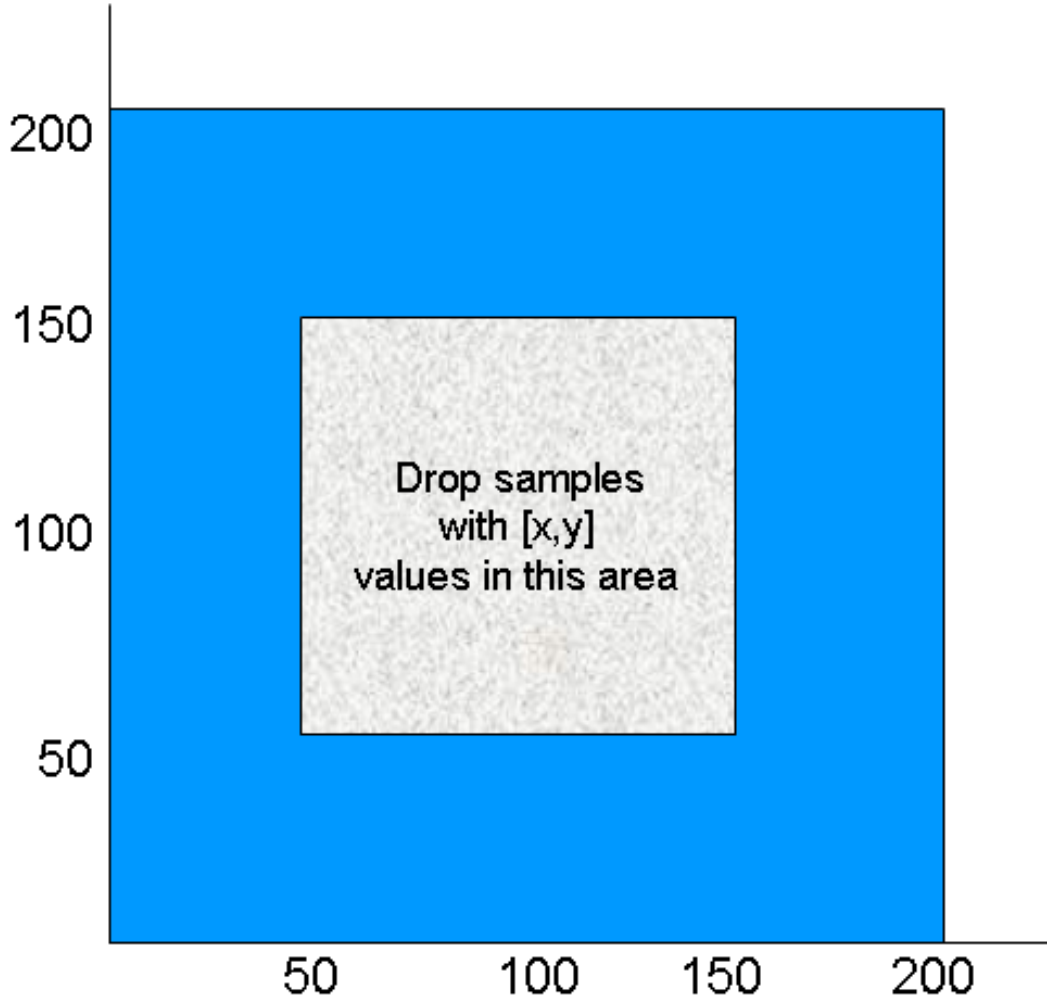



Figure 3.7: Filtering Example

The filter expression would look like this (remember the expression is written so that DDS samples that we *do* want will *pass*):

```
X < 50 OR X > 150 OR Y < 50 OR Y > 150
```

**Note:** This release of *Connext Micro* does not support parentheses in SQL expressions. Operator precedence from highest to lowest is:

1. NOT
2. AND
3. OR

That is, NOT applies to the single predicate that immediately follows it; AND binds tighter than OR.

---

Suppose you would like the ability to adjust the coordinates that are considered outside the acceptable range (changing the size of the gray box). You can achieve this by changing the whole filter expression or by using filter parameters. The expression can be written using filter parameters as follows:

```
X < %0 OR X > %1 OR Y < %2 OR Y > %3
```

See the String and Sequence Support sections of the API Reference for more information.

In C++, the filter parameters could be assigned like this:

```
dr_qos.content_filter.expression_parameters.maximum(4);
dr_qos.content_filter.expression_parameters.length(4);
dr_qos.content_filter.expression_parameters[0] = DDS_String_dup("50");
dr_qos.content_filter.expression_parameters[1] = DDS_String_dup("150");
dr_qos.content_filter.expression_parameters[2] = DDS_String_dup("50");
dr_qos.content_filter.expression_parameters[3] = DDS_String_dup("150");
```

### SQL grammar

This section describes the subset of SQL syntax, in Backus-Naur Form (BNF), that you can use to form filter expressions.

We use the following notational conventions:

- *NonTerminals* are typeset in italics.
- 'Terminals' are quoted and typeset in a fixed-width font. They are written in upper case in most cases in the BNF-grammar below, but should be case insensitive.
- **TOKENS** are typeset in bold.
- The notation (element // ',') represents a non-empty, comma-separated list of elements.

```
FilterExpression ::= Condition
Condition ::= Predicate
              |    Condition 'AND' Condition
              |    Condition 'OR' Condition
              |    'NOT' Condition
Predicate ::= ComparisonPredicate
              |   BetweenPredicate
ComparisonPredicate ::= FIELDNAME RelOp Parameter
                                  |    Parameter RelOp FIELDNAME
```

(continues on next page)

(continued from previous page)

```
                                    |    FIELDNAME RelOp FIELDNAME
BetweenPredicate ::= FIELDNAME 'BETWEEN' Range
                               |   FIELDNAME 'NOT BETWEEN' Range
RelOp ::= '=' | '>' | '>=' | '<' | '<=' | '<>'
Range ::= Parameter 'AND' Parameter
Parameter ::= INTEGERVALUE
                   |   CHARVALUE
                   |   FLOATVALUE
                   |   ENUMERATEDVALUE
                   |   BOOLEANVALUE
                   |   PARAMETER
```

#### Token expressions

This section describes the syntax and meaning of the tokens used in SQL grammar.

**FIELDNAME**: A reference to a field in the data structure. Each identifier in the field name is any series of characters 'a', ..., 'z', 'A', ..., 'Z', '0', ..., '9', '_' but may not start with a digit. Nested fields are referenced using dotted notation (for example, `outer.inner.value`); each component of the path before the final field must be a non-optional struct member. `typedef` aliases to structs, primitives, and enums are resolved transparently. The names of the fields are those specified in the IDL definition of the corresponding structure, which may or may not match the fieldnames that appear on the language-specific mapping of the structure. FIELDNAME must resolve to a primitive IDL type; that is boolean, octet, int16, uint16, int32, uint32, int64, uint64, float, double, char, or enum.

```
FIELDNAME: IDENT ('.' IDENT)*
IDENT: LETTER (PART_LETTER)*
LETTER: [ "A"-"Z","_","a"-"z" ]
PART_LETTER: [ "A"-"Z","_","a"-"z","0"-"9" ]
```

Primitive IDL types referenced by FIELDNAME are treated as different types in predicate according to the following table:

| Predicate Data Type | IDL Type |
|---|---|
| BOOLEANVALUE | boolean |
| INTEGERVALUE | octet, int16, uint16, int32, uint32, int64, uint64 |
| FLOATVALUE | float, double |
| CHARVALUE | char |
| ENUMERATEDVALUE | enum |

**INTEGERVALUE**: Any series of digits, optionally preceded by a plus or minus sign, representing a decimal integer value within the range of the system. '**L**' or '**l**' must be used for int64 (long long), otherwise int32 (long) is assumed. A hexadecimal number is preceded by **0x** and must be a valid hexadecimal expression.

```
INTEGERVALUE : (["+","-"])? (["0"-"9"])+ [("L","l")]? | (["+","-"])? ["0x","0X"](["0"-"9
↪", "A"-"F", "a"-"f"])+ [("L","l")]?
```

**CHARVALUE**: A single character enclosed between single quotes.

```
CHARVALUE : "'" (~["'"])? "'"
```

**FLOATVALUE**: Any series of digits, optionally preceded by a plus or minus sign and optionally including a floating point ('**.**'). '**F**' or '**f**' must be used for float, otherwise double is assumed. A power-of-ten expression may be postfixed, which has the syntax en or En, where n is a number, optionally preceded by a plus or minus sign.

```
FLOATVALUE : (["+","-"])? (["0"-"9"])* (".")? (["0"-"9"])+ (EXPONENT)?[("F",'f')]?
```

where EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+

**ENUMERATEDVALUE**: A reference to a value declared within an enumeration. Enumerated values consist of the name of the enumeration label enclosed in single quotes. The name used for the enumeration label must correspond to the label names specified in the IDL definition of the enumeration.

```
ENUMERATEDVALUE : "'" ["A" - "Z", "a" - "z"] ["A" - "Z", "a" - "z", "_", "0" - "9"]* "'"
```

**BOOLEANVALUE**: Can either be **TRUE** or **FALSE**, and is case insensitive.

```
BOOLEANVALUE : ["TRUE","FALSE"]
```

**PARAMETER**: Takes the form %:italic:*n*, where n represents a natural number (zero included) smaller than 100. It refers to the (n + 1)th argument in the given context. This argument can only be in primitive type value format. It cannot be a FIELDNAME.

```
PARAMETER : "%" (["0"-"9"])+
```

#### Type compatibility in the predicate

As seen in Table 3.8, only certain combinations of type comparisons are valid in the predicate.

Table 3.8: Valid Type Comparisons

| | Boolean | Integer | Float | Char | Enum |
|---|---|---|---|---|---|
| Boolean | ✓[3] | | | | |
| Integer | | ✓ | ✓ | | ✓[1] |
| Float | | ✓ | ✓ | | |
| Char | | | | ✓ | [2] |
| Enum | | ✓[1] | | [2] | ✓ |

[3] Boolean values may only be compared using `=` or `<>`. The relational operators (`<`, `<=`, `>`, `>=`) and `BETWEEN` / `NOT BETWEEN` are not valid for boolean operands.

[1] An integer variable cannot be compared with an enumerated constant. For example, `MyInteger='EnumValue'`

#### Enumerations

A filter expression can use enumeration values, such as GREEN, instead of the numerical value. For example, if **x** is an enumeration of GREEN, YELLOW and RED, the following expressions are valid:

```
"x = 'GREEN'"
"X < 'RED'"
```

### 3.9.5 Custom filters

In addition to the built-in SQL filter, you can implement your own content filter by providing a DDS_ContentFilterI interface. A custom filter is useful when you need domain-specific filtering logic that cannot be expressed in SQL, or when you want full control over how filtering resources are managed.

One reason to use a custom filter is that the default filter can only filter based on relational operations between topic members, not on a computation involving topic members. For example, if you want to filter based on the sum of the members, you must create your own filter.

Another potential reason to create a custom filter is to provide a SQL-compatible filter that is optimized for a specific subset of SQL syntax and expressions. Such an implementation could be more efficient than the built-in SQL filter, which must support a wide variety of SQL expressions. A filter implementation only must support the expressions that are used by readers in that application. A *DataWriter* which is unable to understand a given filter expression will simply fall back to reader-side filtering for that specific expression.

---

**Note:** RTI Code Generator can generate the examples `custom_filter` and `cert_custom_filter`, which demonstrate a complete custom filter implementation. For more details, refer to *Example Applications*.

---

#### Registering a custom filter

Before a custom filter can be used by a *DataReader* (or by a *DataWriter* applying writer-side filtering), it must be registered with their parent *DomainParticipant*.

A content filter implementation is identified by a `filter_class_name`. This name must be unique within the *DomainParticipant*. The same name must be used in the *DataReader*'s DDS_ContentFilterQosPolicy to specify that the *DataReader* is using that filter implementation. It must also match the name used to register the filter with the *DomainParticipant* of any *DataWriter* that you want to apply writer-side filtering.

is not valid.

[2] Whether or not a constant value in single quotes is interpreted as a char or an enum depends on the type of the variable it is being compared with. For example, in `MyChar='A'`, A is interpreted as a char constant. In `MyEnum='A'`, A is interpreted as an enum value and must be a defined label for that enumerated type.

The built-in SQL filter, when registered automatically, uses the filter class name `DDS_SQL_CONTENT_FILTER_CLASS`. If you disable the built-in SQL filter, then you can use that name for the custom filter if it provides SQL compatibility.

Follow these steps to register and use a custom filter:

1. **Configure the DomainParticipant QoS:** The default `filter_class_max_count` is `1`, which only accommodates the built-in SQL filter. You must increase it to support additional custom filter classes. You may also need to increase `filter_class_max_length` if your filter class name is longer than 7 characters (the default maximum). See the following example code:

```
struct DDS_DomainParticipantQos dp_qos = DDS_DomainParticipantQos_INITIALIZER;

/* Allow 2 filter classes: built-in SQL + one custom filter */
dp_qos.filter.resource_limits.filter_class_max_count = 2;

/* Increase max class name length for custom filter name (default is 7) */
dp_qos.filter.resource_limits.filter_class_max_length = 15;
```

   If you do not need the built-in SQL filter, you can disable it and set `filter_class_max_count` to `1`:

```
dp_qos.filter.disable_builtin_sql_filter = DDS_BOOLEAN_TRUE;
dp_qos.filter.resource_limits.filter_class_max_count = 1;
```

2. **Create the DomainParticipant:** Use the configured QoS policy above for the created *DomainParticipant*.

3. **Register the custom filter:** Call `DDS_DomainParticipant_register_contentfilter()` to register a new filter implementation with the *DomainParticipant*.

   The `filter_class_name` parameter must be unique within the *DomainParticipant* and must match the `filter_class_name` in the *DataReader*'s DDS_ContentFilterQosPolicy.

   The `filter_intf` parameter is a pointer to a DDS_ContentFilterI struct that contains the callback functions to implement the filter's behavior. See *Implementing DDS_ContentFilterI* for more details on the required callbacks.

   The `filter_property` parameter is an optional user-defined pointer that is passed to the filter's `create_instance` callback. It can be used to pass configuration information or other data to the filter when it is being created. See the following example code:

```
DDS_ReturnCode_t retcode;

retcode = DDS_DomainParticipant_register_contentfilter(
        participant,
        "MY_FILTER_CLASS",
        MyFilterClass_get_interface(),
        &my_prop /* optional user property */);

if (retcode != DDS_RETCODE_OK)
{
```

(continues on next page)

(continued from previous page)

```
    /* Error handling */
}
```

4. **Create a DataReader with the custom filter class:** Set the `filter_class_name` in the DDS_ContentFilterQosPolicy to match the name used during registration, as shown here:

```
struct DDS_DataReaderQos dr_qos = DDS_DataReaderQos_INITIALIZER;

dr_qos.content_filter.filter_name = DDS_String_dup("MyFilter");
dr_qos.content_filter.filter_class_name = DDS_String_dup("MY_FILTER_CLASS");
dr_qos.content_filter.filter_expression = DDS_String_dup("10 50");

/* ... create the DataReader with this QoS ... */
```

For writer-side filtering to work, the same custom filter must be registered on both the publishing and subscribing *DomainParticipants*. See *Where filtering is applied* for the full set of requirements.

### Unregistering a custom filter

To unregister a filter from a *DomainParticipant*, use the DDS_DomainParticipant_unregister_contentfilter() function. This will free the filter resources by calling the filter's `delete_instance` callback and will remove the filter from the *DomainParticipant*, ensuring it can no longer be used by any entities and allowing the filter class name to be reused for a different filter.

If you attempt to unregister a filter that is still in use by any *DataReader* or *DataWriter*, the unregister function will fail and return `DDS_RETCODE_PRECONDITION_NOT_MET`.

---

**Note:** You don't need to unregister a filter before deleting a *DomainParticipant*. All registered filters will be automatically unregistered and their resources freed when the *DomainParticipant* is deleted. If you don't need to reuse the filter class name, you don't need to manually unregister the filter.

---

### Implementing DDS_ContentFilterI

The DDS_ContentFilterI structure defines five callback functions that you must implement. These callbacks manage the lifecycle of your filter and perform the actual filtering logic:

Table 3.9: DDS_ContentFilterI Callbacks

| Callback | Description |
|---|---|
| `create_instance` | Allocate and initialize filter resources. Called once when the filter is registered with a *DomainParticipant*. Receives a `DDS_FilterResourceLimits` pointer and an optional user-defined property pointer. Returns a pointer to the created filter instance, or `NULL` on failure. |
| `delete_instance` | Free filter resources. Called when the filter is unregistered from a *DomainParticipant*. Receives the filter instance pointer returned by `create_instance` when the filter was registered. |
| `compile` | Parse a filter expression and its parameters into a compiled form. Receives the filter instance, an output parameter `compiled_expression_out` for the compiled result, the expression string, the sequence of string parameters, a `DDS_TypeCode` for type introspection, and the registered type name of the data type. Must set `*compiled_expression_out` to a pointer to the compiled filter state and return `DDS_BOOLEAN_TRUE` on success.<br>The filter expression from the DDS_ContentFilterQosPolicy is passed as-is to this callback's `expression` parameter (it is opaque to *Connext Micro*), so custom filters can define any expression syntax. The `type_class_name` parameter provides the registered type name of the data type. The `type_code` parameter may be `NULL` if XTypes is not enabled. |
| `evaluate` | Evaluate a data sample against a compiled filter. Receives the filter instance, the compiled filter data, and a pointer to the sample. Sets `*result_out` to indicate whether the sample passes the filter. Returns `DDS_BOOLEAN_TRUE` on success.<br>If `evaluate` returns `DDS_BOOLEAN_FALSE` (indicating an error, not a non-matching sample), the sample is dropped on the reader side; on the writer side, the sample is sent without a filter result and the reader will apply its own filter. For samples that simply do not match, set `*result_out` to `DDS_BOOLEAN_FALSE` and return `DDS_BOOLEAN_TRUE`. |
| `finalize` | Free compiled filter resources. Receives the filter instance and the compiled filter data pointer returned by `compile`. Called when the compiled filter is no longer needed, for example when a *DataReader* is deleted or when its filter expression is updated. |

### Threading model

All DDS_ContentFilterI callbacks are invoked while the middleware holds an internal lock on the *DomainParticipant*. This means:

- Callbacks for a given filter instance are serialized by the middleware. A filter implementation does not need to provide its own synchronization for access to a single filter instance.
- If the same filter class is registered on multiple *DomainParticipants*, each receives its own filter instance via a separate call to `create_instance`. Callbacks for different filter instances may execute concurrently.
- Callbacks should complete quickly. Long-running operations in any callback will delay data reception and transmission for other entities on the same *DomainParticipant*.

### Preallocating compiled filter resources

The DDS_ContentFilterI interface is designed to allow the pre-allocation of filter resources and avoid dynamic memory allocation at runtime. The `create_instance` callback receives a `DDS_FilterResourceLimits` struct whose `filter_expression_max_count` field indicates how many compiled filter expressions may be active simultaneously. This value, along with the other resource limits, can be used to pre-allocate all memory that the filter will need when an instance is created instead of allocating memory in the `compile` callback.

The following example shows one possible way to implement a filter that pre-allocates compiled filter resources in the `create_instance` callback:

```
struct MyCompiledFilter
{
    DDS_Boolean in_use;
    /* ... other fields for compiled filter state ... */
};

struct MyFilterClass
{
    struct MyCompiledFilter *compiled_pool;
    RTI_UINT32 pool_size;
};

/* create_instance: allocate a pool of compiled filter entries */
void *
MyFilterClass_create_instance(
        const struct DDS_FilterResourceLimits *resource_limits,
        const void *property)
{
    struct MyFilterClass *filter = NULL;

    OSAPI_Heap_allocate_struct(&filter, struct MyFilterClass);
    if (filter == NULL)
    {
        return NULL;
```

(continues on next page)

(continued from previous page)

```
    }

    filter->pool_size = (RTI_UINT32)resource_limits->filter_expression_max_count;

    OSAPI_Heap_allocate_array(&filter->compiled_pool,
                              filter->pool_size,
                              struct MyCompiledFilter);
    if (filter->compiled_pool == NULL)
    {
        OSAPI_Heap_free_struct(filter);
        return NULL;
    }

    return filter;
}

/* compile: find an unused pool entry instead of allocating */
DDS_Boolean
MyFilterClass_compile(
        void *filter_instance,
        void **compiled_expression_out,
        const char *expression,
        const struct DDS_StringSeq *parameters,
        const struct DDS_TypeCode *type_code,
        const char *type_class_name)
{
    struct MyFilterClass *filter = (struct MyFilterClass *)filter_instance;
    struct MyCompiledFilter *compiled = NULL;

    /* Find an unused entry in the pool */
    for (RTI_UINT32 i = 0; i < filter->pool_size; i++)
    {
        if (!filter->compiled_pool[i].in_use)
        {
            compiled = &filter->compiled_pool[i];
            break;
        }
    }

    if (compiled == NULL)
    {
        /* Pool exhausted */
        return DDS_BOOLEAN_FALSE;
    }

    compiled->in_use = DDS_BOOLEAN_TRUE;

    /* ... parse expression and parameters ... */

    *compiled_expression_out = compiled;
    return DDS_BOOLEAN_TRUE;
```

(continues on next page)

(continued from previous page)

```
}

/* evaluate: apply the compiled filter to a sample */
DDS_Boolean
MyFilterClass_evaluate(
        void *filter_instance,
        void *compiled_expression,
        const void *sample,
        DDS_Boolean *result_out)
{
    struct MyCompiledFilter *compiled = (struct MyCompiledFilter *)compiled_expression;

    /* ... evaluate the sample against the compiled filter ... */

    *result_out = DDS_BOOLEAN_TRUE;
    return DDS_BOOLEAN_TRUE;
}

/* finalize: return the entry to the pool by clearing the flag */
void
MyFilterClass_finalize(
        void *filter_instance,
        void *compiled_expression)
{
    struct MyCompiledFilter *compiled = (struct MyCompiledFilter *)compiled_expression;

    compiled->in_use = DDS_BOOLEAN_FALSE;
}

/* delete_instance: free the pool and filter struct (non-CERT only) */
void
MyFilterClass_delete_instance(void *filter_instance)
{
    struct MyFilterClass *filter = (struct MyFilterClass *)filter_instance;

    OSAPI_Heap_free_array(filter->compiled_pool);
    OSAPI_Heap_free_struct(filter);
}

/* static content filter interface instance */
static struct DDS_ContentFilterI MyFilterClass_fv_interface =
{
    MyFilterClass_create_instance,
#ifndef RTI_CERT
    MyFilterClass_delete_instance,
#endif
    MyFilterClass_compile,
    MyFilterClass_evaluate,
    MyFilterClass_finalize
};
```

(continues on next page)

(continued from previous page)

```
/* get_interface: return a pointer to the filter interface */
struct DDS_ContentFilterI *
MyFilterClass_get_interface(void)
{
    return &MyFilterClass_fv_interface;
}
```

## 3.10 Generating Type Support with rtiddsgen

### 3.10.1 Why Use rtiddsgen?

For *Connext Micro* to publish and subscribe to topics of user-defined data types, the types must be defined and programmatically registered with *Connext Micro*. A registered type is then serialized and deserialized by *Connext Micro* through a pluggable interface that each type must implement.

Rather than manually implementing each new type yourself, *Connext Micro* provides you with the *rtiddsgen* IDL compiler to generate type support code.

### 3.10.2 IDL Type Definition

*rtiddsgen* for *Connext Micro* accepts types defined in IDL. The HelloWorld examples included with *Connext Micro* use the following HelloWorld.idl:

```
struct HelloWorld
{
    string<128> msg;
};
```

For further reference, see the section on Creating User Data Types with IDL in the RTI Connext DDS Core Libraries User's Manual.

### 3.10.3 Generating Type Support

Before running *rtiddsgen*, you must set some environment variables:

- `RTIMEHOME` sets the path to the *Connext Micro* installation directory.
- `JREHOME` sets the path for a Java JRE.

> **Note:** A JRE is shipped with *Connext Professional* on platforms supported for the execution of rtiddsgen (Linux®, Windows®, and macOS®). It is not necessary to set `JREHOME` on these platforms, unless a specific JRE is preferred.

### Generating type support for C

Run *rtiddsgen* from the command line to generate C language type support for a UserType.idl (and replace any existing generated files):

```
> $rti_connext_micro_root/bin/rtiddsgen -micro -language C -replace UserType.idl
```

### Generating type support for C++

Run *rtiddsgen* from the command line to generate C++ language type-support for a UserType.idl (and replace any existing generated files):

```
> $rti_connext_micro_root/bin/rtiddsgen -micro -language C++ -replace UserType.idl
```

**Note:** *Connext Micro* only supports traditional C++.

### Notes on command line options

In order to generate type support for *Connext Micro* with *rtiddsgen*, you must specify the `-micro` option on the command line.

To list all command-line options specifically supported by *rtiddsgen* for *Connext Micro*, enter:

```
> rtiddsgen -micro -help
```

**Note:** You might notice that the previously available options `-language microC` and `-language microC++` have been replaced with `-micro -language C` and `-micro -language C++`, respectively. It is still possible to specify `microC` and `microC++` for backwards compatibility, but we advise switching to using the `-micro` command-line option along with other arguments.

### Generated type support files

*rtiddsgen* will produce the following header and source files for each IDL file passed to it:

- `UserType.h` and `UserType.c` (`.cxx` for C++) implement creation/initialization and deletion (only for *Connext Micro*) of a single sample and a sequence of samples of the type (or types) defined in the IDL description.
- `UserTypePlugin.h` and `UserTypePlugin.c` (`.cxx` for C++) implement the pluggable type interface that *Connext Micro* uses to serialize and deserialize the type.
- `UserTypeSupport.h` and `UserTypeSupport.c` (`.cxx` for C++) define type-specific *DataWriters* and *DataReaders* for user-defined types.

### 3.10.4 Using Custom Types in Connext Micro Applications

A *Connext Micro* application must include the generated headers and register the type with the *DomainParticipant* before a topic of that type can be created. For an example HelloWorld type, the following code registers the type with the participant and then creates a topic of that type:

```
#include "HelloWorld.h"
#include "HelloWorldPlugin.h"
#include "HelloWorldSupport.h"

/* ... */

retcode = HelloWorldTypeSupport_register_type(application->participant,
                                              "HelloWorld");
if (retcode != DDS_RETCODE_OK)
{
    /* Log an error */
    goto done;
}

application->topic =
    DDS_DomainParticipant_create_topic(application->participant,
                                       "Example HelloWorld",
                                       "HelloWorld",
                                       &DDS_TOPIC_QOS_DEFAULT, NULL,
                                       DDS_STATUS_MASK_NONE);

if (application->topic == NULL)
{
    /* Log an error */
    goto done;
}
```

The generated source files must be compiled and linked with the application. See the full HelloWorld examples for further reference.

### 3.10.5 Customizing Generated Code

*rtiddsgen* allows you to select whether you want to generate code to subscribe to and/or publish a custom data type. When generating code for subscriptions, only those parts of code dealing with deserialization of data and the implementation of a typed *DataReader* endpoint are generated. Conversely, only those parts of code addressing serialization and the implementation of a *DataWriter* are considered when generating publishing code.

Control over these options is provided by two command-line arguments:

- `-reader` generates code for deserializing custom data-types and creating *DataReaders* for the type.
- `-writer` generates code for serializing custom data-types and creating *DataWriters* for the type.

If neither of these two options are supplied to *rtiddsgen*, both *DataReaders* and *DataWriters* will be generated. If both options are supplied, both are enabled.

### 3.10.6 Unsupported Features of rtiddsgen with Connext Micro

*Connext Micro* supports a subset of the features and options in *rtiddsgen*. Use `rtiddsgen -micro -help` to see the list of features supported by *rtiddsgen* for *Connext Micro*.

## 3.11 Threading Model

### 3.11.1 Introduction

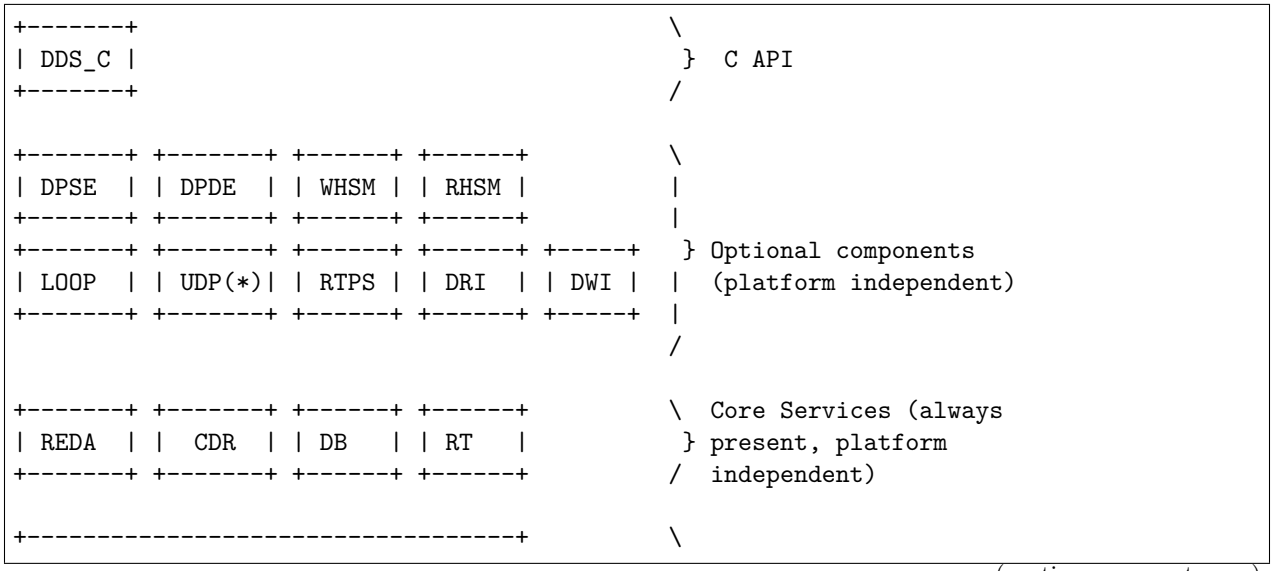
This section describes the threading model, the use of critical sections, and how to configure thread parameters in *RTI Connext Micro*. Please note that the information contained in this document applies to application development using *Connext Micro*.

This section includes:

- *Architectural Overview*
- *Threading Model*
- *UDP Transport Threads*

### 3.11.2 Architectural Overview

*RTI Connext Micro* consists of a core library and a number of components. The core library provides a porting layer, frequently used data-structures and abstractions, and the DDS API. Components provide additional functionality such as UDP communication, DDS discovery plugins, DDS history caches, etc.




(continues on next page)

(continued from previous page)

```
|               OSAPI                |           } Platform dependent module
+------------------------------------+          /

(*) The UDP transport relies on a BSD socket API
```

### 3.11.3 Threading Model

*RTI Connext Micro* is architected in a way that makes it possible to create a port of *Connext Micro* that uses no threads, for example on platforms with no operating system or where the OS itself creates the threads which calls into *Connext Micro*. Thus, the following discussion is only applicable to *Connext Micro* libraries from RTI that runs on operating systems where *Connext Micro* is allowed to create threads. The platform specific notes provide additional information regarding integration with specific environments.

#### OSAPI Threads

The *Connext Micro* OSAPI layer creates one thread per OS process. This thread manages all the *Connext Micro* timers, such as deadline and liveliness timers. This thread is created by the *Connext Micro* OSAPI System when the OSAPI_System_initialize() function is called. When the *Connext Micro* DDS API is used DomainParticipantFactory_get_instance() calls this function once.

#### Configuring OSAPI Threads

The timer thread is configured through the OSAPI_SystemProperty structure and any changes must be made before OSAPI_System_initialize() is called. In *Connext Micro*, DomainParticipantFactory_get_instance() calls OSAPI_System_initialize(). Thus, if it is necessary to change the system timer thread settings, it must be done before DomainParticipantFactory_get_instance() is called the first time.

Please refer to OSAPI_Thread for supported thread options. Note that not all options are supported by all platforms.

```
struct OSAPI_SystemProperty sys_property = OSAPI_SystemProperty_INITIALIZER;

if (!OSAPI_System_get_property(&sys_property))
{
    /* ERROR */
}

/* Please refer to OSAPI_ThreadOptions for possible options */
sys.property.timer_property.thread.options = ....;

/* The stack-size is platform dependent, it is passed directly to the OS */
sys.property.timer_property.thread.stack_size = ....
```

(continues on next page)

(continued from previous page)

```
/* The priority is platform dependent, it is passed directly to the OS */
sys.property.timer_property.thread.priority = ....

if (!OSAPI_System_set_property(&sys_property))
{
    /* ERROR */
}
```

### UDP Transport Threads

Of the components that RTI provides, only the UDP component creates threads. The UDP transport creates one receive thread for each unique UDP receive address and port. Thus, three UDP threads are created by default:

- A multicast receive thread for discovery data (assuming multicast is available and enabled)
- A unicast receive thread for discovery data
- A unicast receive thread for user-data

Additional threads may be created depending on the transport configuration for a *DomainParticipant*, *DataReader* and *DataWriter*. The UDP transport creates threads based on the following criteria:

- Each unique unicast port creates a new thread
- Each unique multicast address *and* port creates a new thread

For example, if a *DataReader* specifies its own multicast receive address a new receive thread will be created.

### Configuring UDP Receive Threads

All threads in the UDP transport share the same thread settings. It is important to note that all the UDP properties must be set before the UDP transport is registered. *Connext Micro* pre-registers the UDP transport with default settings when the DomainParticipantFactory is initialized. To change the UDP thread settings, use the following code.

```
RT_Registry_T *registry = NULL;
DDS_DomainParticipantFactory *factory = NULL;
struct UDP_InterfaceFactoryProperty *udp_property = NULL;

factory = DDS_DomainParticipantFactory_get_instance();

udp_property = (struct UDP_InterfaceFactoryProperty *)
                    malloc(sizeof(struct UDP_InterfaceFactoryProperty));
*udp_property = UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;

registry = DDS_DomainParticipantFactory_get_registry(factory);
```

(continues on next page)

(continued from previous page)

```
if (!RT_Registry_unregister(registry, "_udp", NULL, NULL))
{
    /* ERROR */
}

/* Please refer to OSAPI_ThreadOptions for possible options */
udp_property->recv_thread.options = ...;

/* The stack-size is platform dependent, it is passed directly to the OS */
udp_property->recv_thread.stack_size = ....

/* The priority is platform dependent, it is passed directly to the OS */
udp_property->recv_thread.priority = ....

if (!RT_Registry_register(registry, "_udp",
                          UDP_InterfaceFactory_get_interface(),
                          (struct RT_ComponentFactoryProperty*)udp_property,
                          NULL))
{
    /* ERROR */
}
```

#### General Thread Configuration

The *Connext Micro* architecture consists of a number of components and layers, and each layer and component has its own properties. It is important to remember that the layers and components are configured independently of each other, as opposed to configuring everything through DDS. This design makes it possible to relatively easily swap out one part of the library for another.

All threads created based on *Connext Micro* OSAPI APIs use the same OSAPI_ThreadProperty structure.

### 3.11.4 Thread Safety

All public APIs have a note about thread safety included in the API reference manuals. It is important that an application does not violate thread safety guidelines.

*RTI Connext Micro* may create multiple threads, but from an application point of view, there is only a single critical section protecting all DDS resources within a *DomainParticipant*.

---

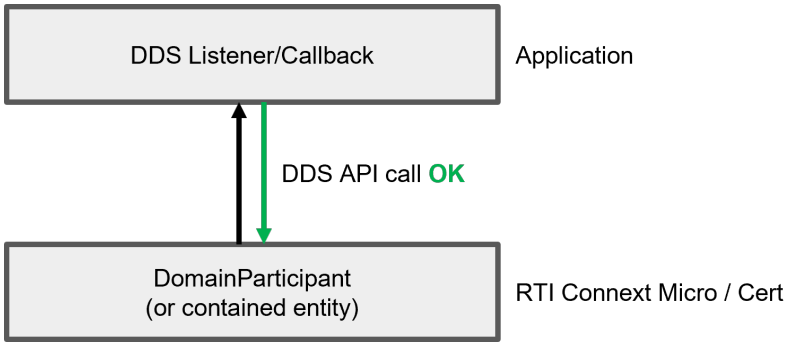
**Note:** Although *Connext Micro* may create multiple mutexes, these are used to protect resources in the OSAPI layer, and are thus not relevant when using the public DDS APIs.

---

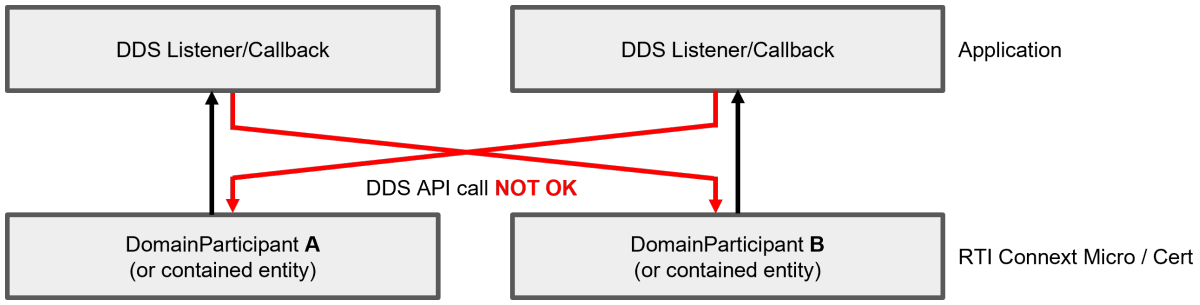
### Calling DDS APIs from listeners and callbacks

When DDS is executing in a listener, it holds a critical section. Thus it is important to return as quickly as possible to avoid stalling network I/O.

There are no deadlock scenarios when a listener calls *Connext Micro* DDS APIs from the **same** *DomainParticipant* (and contained entities) that the listener was called from, as shown in the diagram below:




**Warning:** It is **not** safe to call DDS APIs from a **different** *DomainParticipant* than the one listener was called from, as shown in the diagram below. This may result in a deadlock situation.




**Warning:** There are no checks on whether or not an API call will cause problems, such as deleting a participant when processing data in on_data_available from a reader within the same participant.

#### Calling DDS APIs from a type-plugin

A user type-plugin that is registered with the *DomainParticipant* is subject to the following rules:

- The key kind is constant.
- The plugin is constant for a given DDS entity (*Topic*, *DataWriter*, or *DataReader*).
- The plugin data must be protected if thread safety is a concern, as it is user data and not protected by *Connext Micro*.

---

**Note:** A type-plugin generated from an IDL file with the *rtiddsgen* IDL compiler included with *Connext Micro* will satisfy these rules.

---

### 3.11.5 Critical Sections

*RTI Connext Micro* may create multiple threads, but from an application point of view there is only a single critical section protecting all DDS resources. Note that although *Connext Micro* may create multiple mutexes, these are used to protect resources in the OSAPI layer and are thus not relevant when using the public DDS APIs.

#### Calling DDS APIs from listeners

When DDS is executing in a listener, it holds a critical section. Thus it is important to return as quickly as possible to avoid stalling network I/O.

There are no deadlock scenarios when calling *Connext Micro* DDS APIs from a listener. However, there are no checks on whether or not an API call will cause problems, such as deleting a participant when processing data in on_data_available from a reader within the same participant.

## 3.12 Batching

This section is organized as follows:

- *Overview*
- *Interoperability*
- *Performance*
- *Example Configuration*

### 3.12.1 Overview

Batching refers to a mechanism that allows *RTI Connext Micro* to collect multiple user data DDS samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

*Connext Micro* supports receiving batches of user data DDS samples, but does not support any mechanism to collect and send batches of user data.

Receiving batches of user samples is transparent to the application, which receives the samples as if the samples had been received one at a time. Note though that the reception sequence number refers to the sample sequence number, not the RTPS sequence number used to send RTPS messages. The RTPS sequence number is the batch sequence number for the entire batch.

A *Connext Micro DataReader* can receive both batched and non-batched samples.

For a more detailed explanation, please refer to the BATCH QosPolicy section in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

### 3.12.2 Interoperability

*RTI Connext Professional* supports both sending and receiving batches, whereas *RTI Connext Micro* supports only receiving batches. Thus, this feature primarily exists in *Connext Micro* to interoperate with *RTI Connext* applications that have enabled batching. An *Connext Micro DataReader* can receive both batched and non-batched samples.

### 3.12.3 Performance

The purpose of batching is to increase throughput when writing small DDS samples at a high rate. In such cases, throughput can be increased several-fold, approaching much more closely the physical limitations of the underlying network transport.

However, collecting DDS samples into a batch implies that they are not sent on the network immediately when the application writes them; this can potentially increase latency. But, if the application sends data faster than the network can support, an increased proportion of the network's available bandwidth will be spent on acknowledgements and DDS sample resends. In this case, reducing that overhead by turning on batching could decrease latency while increasing throughput.

### 3.12.4 Example Configuration

This section includes several examples that explain how to enable batching in *RTI Connext Professional*. For more detailed and advanced configuration, please refer to the RTI Connext DDS Core Libraries User's Manual.

- This configuration ensures that a batch will be sent with a maximum of 10 samples:

```
<datawriter_qos>
    <publication_name>
        <name>HelloWorldDataWriter</name>
    </publication_name>
    <batch>
        <enable>true</enable>
        <max_samples>10</max_samples>
    </batch>
</datawriter_qos>
```

- This configuration ensures that a batch is automatically flushed after the delay specified by max_flush_delay. The delay is measured from the time the first sample in the batch is written by the application:

```
<datawriter_qos>
    <publication_name>
        <name>HelloWorldDataWriter</name>
    </publication_name>
    <batch>
        <enable>true</enable>
        <max_flush_delay>
            <sec>1</sec>
            <nanosec>0</nanosec>
        </max_flush_delay>
    </batch>
</datawriter_qos>
```

- The following configuration ensures that a batch is flushed automatically when max_data_bytes is reached (in this example 8192).

```
<datawriter_qos>
    <publication_name>
        <name>HelloWorldDataWriter</name>
    </publication_name>
    <batch>
        <enable>true</enable>
        <max_data_bytes>8192</max_data_bytes>
    </batch>
</datawriter_qos>
```

Note that max_data_bytes does not include the metadata associated with the batch samples.

Batches must contain whole samples. If a new batch is started and its initial sample causes the serialized size to exceed max_data_bytes, *RTI Connext Professional* will send the sample in a single batch.

## 3.13 Message Integrity Checking

*Connext Micro* uses the DDS-I/RTPS protocol for communication between DDS applications, and RTPS messages are sent and received by a transport. When an RTPS message is sent across a communication link, such as Ethernet, it is possible that some bits may change value. These errors may cause communication failures or incorrect data to be received. In order to *detect* these types of errors, transports such as UDP often include a checksum to validate the integrity of the data: a sender adds the checksum to the transmitted data and the receiver validates that the calculated checksum for the received data matches the checksum received from the sender. If the checksums are different, a data corruption has occurred.

By default, *Connext Micro* relies on the underlying transport, such as UDP, to handle data integrity checking. However, the underlying transport may not provide sufficient integrity checking, or may itself introduce errors that *Connext Micro* must be able to detect regardless of the transport.

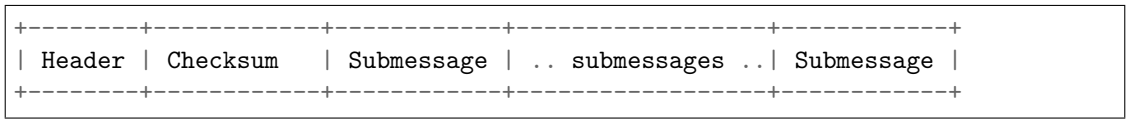
In order to address both of these scenarios for *any* transport, *Connext Micro* supports RTPS message integrity checking by adding a checksum to the RTPS message itself. This chapter describes the setup and default options to access this feature.

For information on how to write custom checksum functions, please refer to *RTPS*.

### 3.13.1 RTPS checksum

*Connext Micro* implements checksum validation on a complete RTPS message. A typical RTPS message without a checksum has the following structure:

When the message integrity checking feature is enabled, the structure of the RTPS message changes as illustrated below:




The sender calculates the checksum for the entire message with a checksum field set to 0 and places the result in the checksum field.

The receiver saves the the received checksum, sets the received checksum field to 0, and calculates the checksum for the entire message. It then compares the calculated checksum with the received checksum. If the checksums differ, the entire RTPS message is considered corrupted.

Note that the checksum is used *only* for error *detection* and *not* for error *correction*.

### 3.13.2 Configurations

You can configure your application to define which algorithms to use and validate as well as the requirements enforced by the participant when communicating with other participants using the DDS_WireProtocolQosPolicy.

Configuring the message integrity checking consists of the two parts:

1. Selecting the checksum algorithm.
2. Configuring how a participant applies the checksums.

#### Selecting a checksum algorithm

*Connext Micro* supports three built-in algorithms and can be configured to use any of the following algorithms:

1. DDS_CHECKSUM_BUILTIN32: CRC-32 As defined by ISO/IEC 13239:2002.
2. DDS_CHECKSUM_BUILTIN64: CRC-64 As defined by ISO/IEC 13239:2002.
3. DDS_CRC_BUILTIN128: MD5 Message Digest

The CRC functions have the following properties:

| Checksum | Polynom | Initial Value | Input Reflected | Output Reflected | XOR Value |
|---|---|---|---|---|---|
| CRC-32 | 0x04c11db7 | 2^32 - 1 | true | true | 2^32 - 1 |
| CRC-64 | 0x1b | 2^64 - 1 | true | true | 2^64 -1 |

Please refer to *RTPS* for information on how to implement custom checksum functions.

#### Configuring the DDS DomainParticipant

The RTPS message integrity feature is configured in the DDS_WireProtocolQosPolicy for a participant. This QoS determines which RTPS checksum should be allowed, and if checksums should be sent and/or validated.

The following three fields determine how a participant uses RTPS checksums:

- `compute_crc` - This configures the participant to send a checksum with each RTPS message. Which checksum to send is determined by `computed_crc_kind`.
- `check_crc` - This configures the participant to verify the checksum in each received RTPS message if the checksum is present. If the checksum is valid, the message is accepted; otherwise, the message is dropped. If a message is received *without* a checksum, it is accepted and processed.
- `require_crc` - This configures the participant to *require* that a checksum is present in the receiving packet. Messages without a checksum are dropped without further processing. Note that this option is orthogonal to the `check_crc` options. This option only requires that a

checksum is included, it does not validate it. To validate and only accept messages with a checksum, *both* check_crc and require_crc must be `true`.

The following two fields determine which checksums are used:

- `computed_crc_kind` - The checksum type to include in each RTPS message when `compute_crc` is `true`.
- `allowed_crc_mask` - A mask of all checksum algorithms that the participant can verify. This allows the participant to receive messages from other participants with a different `computed_crc_kind`. A participant will ignore a participant that is sending a checksum that it cannot validate.

For example, the following snippet shows how to configure the participant to:

- Send all messages (except the participant announcements; see *Participant discovery and compatibility* below) with `DDS_CHECKSUM_BUILTIN64`.
- Accept `DDS_CHECKSUM_BUILTIN32`, `DDS_CHECKSUM_BUILTIN64`, and `DDS_CHECKSUM_BUILTIN128` algorithms.

```
struct DDS_DomainParticipantQos dp_qos =
                                    DDS_DomainParticipantQos_INITIALIZER;

dp_qos.protocol.computed_crc_kind =  DDS_CHECKSUM_BUILTIN64;

dp_qos.protocol.allowed_crc_mask = DDS_CHECKSUM_BUILTIN32
                                    | DDS_CHECKSUM_BUILTIN64
                                    | DDS_CHECKSUM_BUILTIN128;
```

### 3.13.3 Participant discovery and compatibility

*Connext Micro* ensures that participants establish communication with each other only when they have compatible checksum configurations. If `compute_crc` is `true`, all messages sent from the participant are protected by a checksum. Since each participant can use a different type of checksum, a mechanism is required to ensure that participants are compatible during discovery.

To bootstrap this mechanism, all participant announcements (if `compute_crc` is set to true) include a checksum of type `DDS_CHECKSUM_BUILTIN32`. The participant announcement carries information about the `computed_crc_kind` (the checksum kind used by the participant) and the `allowed_crc_mask` (the checksum kinds understood by the participant), and whether or not the participant requires a checksum for each RTPS message (if `require_crc` is set to true). Please note that messages with `DDS_CHECKSUM_BUILTIN32` checksum are *always* accepted to enable discovering new participants.

For a Participant (A) to match with another Participant (B), the `computed_crc_kind` of Participant (B) must be a strict subset of the `allowed_crc_mask` of Participant (A) and vice versa. If Participant (B) does not send a checksum (`compute_crc` is set to false), it can only match Participant (A) if it does not set `require_crc` to true.

### 3.13.4 Interoperability with Connext Professional

*Connext Micro* supports two different kinds of RTPS submessages for CRC 32-bit checksums:

- The standard CRC 32-bit checksum in the RTPS header extension, as defined by the OMG.
- The legacy CRC 32-bit checksum submessage used by older versions of *Connext Professional*.

*Connext Micro* will understand and accept either kind of received submessage. However, it may be necessary to change the transmit mode of *Connext Micro* to enable interoperability with older versions of *Connext Professional* and allow *Connext Professional* to validate the checksum.

The following two transmit modes are available:

- **RTPS_CRC_TXMODE_OMG**: uses the standard method as defined by the OMG. This is the default mode. The checksums sent by *Connext Micro* may not be understood by older versions of *Connext Professional* and cause *Connext Professional* to treat the message as if it does not include a checksum.
- **RTPS_CRC_TXMODE_RTICRC32**: uses the legacy CRC32 mode. This mode sets the `computed_crc_kind` to `DDS_CRC_BUILTIN32`. The checksum sent by *Connext Micro* will be understood by older versions of *Connext Professional*. Use this option only if there is a *Connext Professional* application in your system which requires the legacy CRC 32-bit checksum.

**Warning:** *Connext Professional* does not support the `require_crc` field in the DDS_WireProtocolQosPolicy.

## 3.14 Sending Large Data

*Connext Micro* supports transmission and reception of data types that exceed the maximum message size of a transport. This section describes the behavior and the configuration options.

This section includes:


- *How Sending Large Data Works*
- *Key Properties of Fragmentation*
- *Differences Between Fragmentation and Transport Limits*
- *Components Involved in Sending Large Data*
- *Configuring Large Data Support*
- *When to Choose Synchronous vs Asynchronous Publishing*
- *Complete Example*
- *Limitations*

### 3.14.1 How Sending Large Data Works

*Connext Micro* supports transmission and reception of data samples that exceed the maximum message size of a transport. For example, UDP supports a maximum user payload of 65507 bytes. In order to send samples larger than 65507 bytes, *Connext Micro* must split the sample into multiple UDP payloads.

*Connext Micro* supports transmission and reception of data samples that exceed the maximum message size of a transport. When a write request produces a sample that cannot fit in a single transport payload, the *DataWriter* splits the sample into fragments sized for that transport. Those fragments are placed under the control of the task scheduler and the flow controller, which determine when tokens are available to send data. The scheduler wakes up at the configured rate, the flow controller grants tokens based on the configured policy, and the fragments are sent as bandwidth becomes available. On the receiving side, the *DataReader* collects incoming fragments, reassembles them into a complete serialized sample, deserializes the payload, and provides regular data to the application once the reconstruction succeeds.

### 3.14.2 Key Properties of Fragmentation

When working with large data, the following properties govern runtime behavior:

- Fragmentation is always compiled in but can be influenced through configuration.
- Fragmentation and flow control are tracked per *DataWriter*; *RTI Connext Professional* differs by sharing flow controllers across a publisher.
- Fragmentation is evaluated per sample. Two samples of the same type may fragment differently, and the application never manipulates fragments directly.
- The *DataWriter* chooses the fragment size, so writers of the same type may use different fragment boundaries.
- All fragments must arrive before a sample can be reconstructed. Under best-effort reliability, the loss of any fragment implies the loss of the sample. Under reliability, missing fragments may be resent; if they cannot be recovered, the sample is dropped.
- Rapid calls to DDS `write()` can exhaust *DataWriter* resources while a large sample is still being sent, because buffers are only released after the full sample has been transmitted.

### 3.14.3 Differences Between Fragmentation and Transport Limits

*Connext Micro* fragmentation is distinct from lower-level limits such as transport payloads or operating system buffers. However, fragmentation interacts with these limits as follows:

- *Connext Micro* fragmentation controls how a single logical sample is split across transport payloads.
- Transports or the IP layer may fragment datagrams again (for example, IP fragmentation when UDP messages exceed roughly 1488 bytes).

- The sender's maximum transmit message size defines the largest payload that can be placed on the wire per transport message.
- The sender's transport transmit buffer limit governs how many bytes the transport can queue internally.
- The receiver's maximum message size specifies the largest single payload it can accept from the transport.
- The receiver's transport buffer limit controls how much data can be buffered while fragments arrive.

Understanding each layer's constraint helps diagnose whether fragmentation settings or transport limits are responsible for throttling throughput.

### 3.14.4 Components Involved in Sending Large Data

Large data delivery relies on four cooperating subsystems:

- **DataWriter fragmenter**: Detects when a sample exceeds the configured transport payload size and produces fragments aligned with the writer's fragmentation policy.
- **Task scheduler**: Runs in its own thread, wakes up at the configured clock rate, and drives periodic tasks such as large-data transmissions.
- **Flow controller**: Applies the selected scheduling policy and token bucket settings to determine when queued fragments are eligible to leave the writer.
- **DataReader reassembly buffer**: Tracks incoming fragments per remote writer, reconstructs samples when all fragments arrive, and manages reassembly resources.

These roles are fixed; configuration determines how they operate but not how they cooperate.

### 3.14.5 Configuring Large Data Support

All configuration must be applied before the relevant entities are created.

#### Task scheduler configuration

The task scheduler properties must be configured before creating the *DomainParticipantFactory*:

```
/* Configure task scheduler thread properties */
sys_property.task_scheduler.thread.stack_size = 32768;
sys_property.task_scheduler.thread.options = OSAPI_THREAD_DEFAULT_OPTIONS;
sys_property.task_scheduler.thread.priority = OSAPI_THREAD_PRIORITY_DEFAULT;

/* Set the rate at which periodic tasks run (in nanoseconds) */
/* This affects how frequently fragments are sent */
sys_property.task_scheduler.rate = 10000000; /* 10 milliseconds */

if (!OSAPI_System_set_property(&sys_property))
```

(continues on next page)

(continued from previous page)

```
{
    /* error */
}
```

### Flow controller configuration

Flow controllers shape the rate at which fragments leave the writer. Each configuration selects a scheduling policy (round robin, earliest deadline first, or highest priority first) and defines a token bucket that meters the available bandwidth. The built-in controllers cover common patterns, while custom controllers allow precise tuning for large data workloads.

### Creating custom flow controllers

Applications can create custom flow controllers for both synchronous and asynchronous publishing modes:

```
DDS_FlowControllerProperty_t fc_property = DDS_FlowControllerProperty_t_INITIALIZER;

/* Configure token bucket parameters */
fc_property.token_bucket.period.sec = 0;
fc_property.token_bucket.period.nanosec = 100000000; /* 100 ms */
fc_property.token_bucket.bytes_per_token = 1024; /* 1 KB per token */
fc_property.token_bucket.tokens_added_per_period = 10; /* 10 KB per period */
fc_property.token_bucket.tokens_leaked_per_period = 0;
fc_property.token_bucket.max_tokens = 100; /* Max 100 KB burst */

/* Use Earliest Deadline First scheduling */
fc_property.scheduling_policy = DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY;
```

The custom flow controller can then be associated with a *DataWriter* as seen in the examples section below.

For additional background on built-in flow controller policies, see the FlowControllers section and the ASYNCHRONOUS_PUBLISHER QoSPolicy section in the *RTI Connext Professional* User's Manual.

### Writer resource limits

Fragments occupy the same resource pools as regular samples. Increase `resource_limits.max_samples` to cover fragments waiting to be sent, adjust `max_instances` if multiple keyed instances are active, and make sure `max_samples_per_instance` reflects the maximum number of enqueued samples per key while large data is in-flight.

#### Reader fragment reassembly limits

Large samples require the reader to buffer fragments until reconstruction is complete. Tune `reader_resource_limits.max_fragmented_samples` for the aggregate workload and `max_fragmented_samples_per_remote_writer` to cap per-writer usage. Align the regular `resource_limits` fields with the expected steady-state data set so reconstructed samples can be stored once the fragments converge.

Maintain the relationship:

```
max_fragmented_samples >=
    number_of_remote_writers * max_fragmented_samples_per_remote_writer
```

If reassembly resources are exhausted, incoming fragments are dropped until capacity is available.

### 3.14.6 When to Choose Synchronous vs Asynchronous Publishing

Use **synchronous publishing** when:

- You need to know immediately if the write succeeded.
- Your application can tolerate blocking during `write()` calls.
- You prefer simpler resource management.
- You primarily send small or medium-sized samples.

Use **asynchronous publishing** when:

- You send large fragmented samples that benefit from batching.
- You need `write()` to return quickly.
- You require precise control over bandwidth consumption over time.
- You can provision additional resources for queuing pending samples.

### 3.14.7 Complete Example

The following code fragment shows a complete example of configured large data support:

```
/* 1. Configure task scheduler before creating factory */
struct OSAPI_SystemProperty sys_property = OSAPI_SystemProperty_INITIALIZER;

if (!OSAPI_System_get_property(&sys_property))
{
    /* error */
}

sys_property.task_scheduler.thread.stack_size = 32768;
sys_property.task_scheduler.rate = 10000000; /* 10 ms */
```

(continues on next page)

(continued from previous page)

```
if (!OSAPI_System_set_property(&sys_property))
{
    /* error */
}

factory = DDS_DomainParticipantFactory_get_instance();
participant = DDS_DomainParticipantFactory_create_participant(
    factory, 0, &DDS_PARTICIPANT_QOS_DEFAULT, NULL, 0);

/* 2. Create custom flow controller */
DDS_FlowControllerProperty_t fc_property = DDS_FlowControllerProperty_t_INITIALIZER;
fc_property.token_bucket.period.sec = 0;
fc_property.token_bucket.period.nanosec = 100000000; /* 100 ms */
fc_property.token_bucket.bytes_per_token = 1024;
fc_property.token_bucket.tokens_added_per_period = 10;
fc_property.token_bucket.max_tokens = 100;
fc_property.scheduling_policy = DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY;

DDS_FlowController *fc = DDS_DomainParticipant_create_flowcontroller(
    participant, "MyFlowController", &fc_property);

/* 3. Configure writer to use flow controller */
DDS_DataWriterQos dw_qos = DDS_DataWriterQos_INITIALIZER;
dw_qos.publish_mode.kind = DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS;
dw_qos.publish_mode.flow_controller_name = DDS_String_dup("MyFlowController");
dw_qos.resource_limits.max_samples = 32;

/* 4. Configure reader for fragment reassembly */
DDS_DataReaderQos dr_qos = DDS_DataReaderQos_INITIALIZER;
dr_qos.reader_resource_limits.max_fragmented_samples = 64;
dr_qos.reader_resource_limits.max_fragmented_samples_per_remote_writer = 8;
dr_qos.resource_limits.max_samples = 32;
```

### 3.14.8 Limitations

The following known limitations apply to large data support:

- Fragmentation support is compiled in by default. While it cannot be completely disabled at runtime, its behavior can be shaped through QoS and flow controller configuration.
- Scheduler accuracy depends on the operating system. Actual wake-up rates may vary with load.
- Reader-side reassembly requires preallocated resources. If limits are exceeded, fragments are dropped.
- With best-effort reliability, losing any fragment loses the entire sample. With reliability, missing fragments may be resent, but irrecoverable fragments cause the sample to be dropped.
- Maintain the relationship between `max_fragmented_samples` and `max_fragmented_samples_per_remote_writer` to ensure operation with multiple writers.

## 3.15 Zero Copy Transfer

Zero Copy transfer enables *RTI Connext Micro* to transmit data samples without copying them internally, similar to Zero Copy Transfer Over Shared Memory in *Connext Professional*. This offers several benefits, including higher throughput of user data, reduced latency between DDS endpoints (compared to other transports that send serialized data, such as UDP), and decoupling sample size from latency. This is particularly useful in applications with large sample sizes, such as image or lidar point cloud data.

At a high level, Zero Copy transfer works by a *DataWriter* and *DataReader* accessing the same shared memory; see Figure 3.8 below. A Zero Copy-enabled *DataWriter* creates a structure in a shared memory region and allocates samples from its pool to shared memory. When samples are published by the *DataWriter*, matching *DataReaders* are notified via a transport that new samples are available in shared memory. The *DataReader* then accesses the samples in shared memory using the standard DDS read/take APIs. Note that the *DataReader* and *DataWriter* must be co-located—that is, within the same operating system instance.

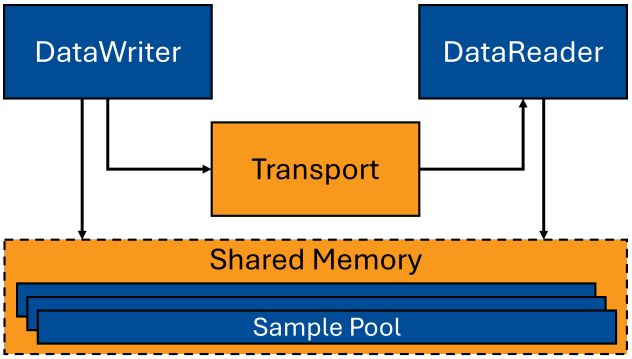



Figure 3.8: Zero Copy Transfer in *Connext Micro*

There are two methods for performing Zero Copy transfer in *Connext Micro*, which we refer to as Zero Copy v1 and v2 in this documentation. The main difference between these two methods is the transport used to notify *DataReaders* that new samples are available:

- Zero Copy v1: *Shared Memory Transport (SHMEM)*
- Zero Copy v2: *Zero Copy v2 Transport*

When choosing between the two versions, consider primarily whether you plan to interoperate *Connext Micro* with *Connext Cert* or *Connext Professional*. Zero Copy v1 is compatible with *Connext Professional*, while Zero Copy v2 is compatible with *Connext Cert*. Other functional differences between v1 and v2 are described in *Compatibility*.

### 3.15.1 Compatibility

Zero Copy v1 and v2 are API-compatible, meaning the API is identical whether you are using the Shared Memory transport or Zero Copy v2 transport. However, there are some important functional differences between the two versions:

1. **Interoperability:** Zero Copy v1 is compatible with the Zero Copy Transfer Over Shared Memory feature in *Connext Professional*, but not with *Connext Cert*. Zero Copy v2 is not compatible with *Connext Professional*, but is compatible with the Zero Copy transfer feature in *Connext Cert*.

   > **Warning:** Zero Copy transfer is not supported on all versions of *Connext Cert*. Consult the documentation in your *Connext Cert* installation for more information.

2. **Sample synchronization:** Zero Copy v2 synchronizes samples between the *DataWriter* and the *DataReader*. This ensures that a *DataReader* cannot access a sample while it is being modified by the *DataWriter*. As a result, the *DataReader* does not need to call the API `FooDataReader_is_data_consistent` to verify whether the sample has been changed, because the API always returns TRUE. In contrast, Zero Copy v1 does not provide any consistency checks, making it necessary for you to call `FooDataReader_is_data_consistent`.

   ---

   **Note:** In Zero Copy v2, a slow *DataReader* may still miss samples if the *DataWriter* overwrites them. These missed samples will not be delivered to the reader. This is similar to Zero Copy v1, where samples can be missed, but v1 offers no guarantees of consistency between *DataWriter* and *DataReader*.

   ---

3. **Volatile DataReaders:** When using Zero Copy v2, a volatile *DataReader* receives all the historical samples available in the *DataWriter*'s queue. Zero Copy v1 uses the `depth` that is set for historical samples.
4. **Sample acknowledgement:** Zero Copy v2 does not support sample acknowledgments. The transport is inherently reliable, meaning that the *DataWriter*'s samples are immediately available to the *DataReader*. However, samples may be removed from the *DataWriter*'s cache when the queue reaches its `history.depth`, even if the reader has not accessed them.
5. **KEEP_ALL support:** Zero Copy v2 does not support the `KEEP_ALL` policy. Zero Copy v1 supports `KEEP_ALL`.
6. **Version priority:** If both Zero Copy v1 and v2 are enabled, *Connext Micro* will prioritize Zero Copy v1 over v2.

The following table summarizes the key differences between the two versions:

Table 3.10: Zero Copy Compatibility

| Feature | Zero Copy v1 | Zero Copy v2 |
|---|---|---|
| Compatibility with *Connext Professional* | ✓ | |
| Compatibility with *Connext Cert* | | ✓[1] |
| Sample synchronization | | ✓ |
| Sample acknowledgement | ✓ | |
| Support for `KEEP_ALL` | ✓ | |
| Transfer of discovery data | ✓[2] | |

### 3.15.2 Overview

Zero Copy samples reside in a shared memory region accessible from multiple processes. When creating a FooDataWriter that supports Zero Copy transfer of user samples, a sample must be created with a new non-DDS API (FooDataWriter_get_loan()). This will return a pointer A* to a sample `Foo` that lies inside a shared memory segment. A reference to this sample will be sent to a receiving FooDataReader across the shared memory. This FooDataReader will attach to a shared memory segment, and a pointer B* to sample `Foo` will be presented to the user. Because the two processes share different memory spaces, A* and B* will be different but they will point to the same place in RAM.

This feature requires using new RTI DDS Extension APIs:

- FooDataWriter_get_loan()
- FooDataWriter_discard_loan()
- FooDataReader_is_data_consistent()

### 3.15.3 Getting started

To enable Zero Copy transfer (either v1 or v2), follow these steps:

1. Annotate your type with the `@transfer_mode(SHMEM_REF)` annotation.

   Currently, variable-length types (strings and sequences) are not supported for types using this transfer mode when a type is annotated with the PLAIN language binding (e.g., `@language_binding(PLAIN)` in IDL).

```
@transfer_mode(SHMEM_REF)
struct HelloWorld
{
    long id;
    char raw_image_data[1024 * 1024]; // 1 MB
};
```

[1] Only if Zero Copy v2 is supported on the corresponding version of *Connext Cert*.
[2] Discovery data is sent over the Shared Memory Transport (SHMEM), but it is not zero copied over.

2. Register the *Shared Memory Transport (SHMEM)* OR the *Zero Copy v2 Transport*. References will be sent across the chosen transport.

   **Note:** If both transports are registered, *Connext Micro* will prioritize the Shared Memory transport (SHMEM) over the Zero Copy v2 transport as described in *Compatibility*.

   **Warning:** If neither transport is registered AND your type is annotated with `@transfer_mode(SHMEM_REF)` (which can occur when using your annotated type with a version of *Connext Micro* that doesn't support Zero Copy), two things will happen:

   1. *Connext Micro* will not create a shared memory region for data samples, and;
   2. Calls to FooDataWriter_get_loan() will fail with `PRECONDITION_NOT_MET`.

   However, the *DataWriter* will still be created and can be used to send samples without using Zero Copy transfer.

3. Create a FooDataWriter for the above type.
4. Get a loan on a sample using FooDataWriter_get_loan().
5. Write a sample using FooDataWriter_write().

For more information, see the example **HelloWorld_zero_copy**, or generate an example for a type annotated with `@transfer_mode(SHMEM_REF)`:

```
rtiddsgen -example -micro -language C HelloWorld.idl
```

### Writing samples

The following code illustrates how to write samples annotated with `@transfer_mode(SHMEM_REF)`:

```
for (int i = 0; i < 10; i++)
{
    Foo* sample;
    DDS_ReturnCode_t dds_rc;
    /* NEW API
       IMPORTANT - call get_loan each time when writing a NEW sample
     */
    dds_rc = FooDataWriter_get_loan(hw_datawriter, &sample);

    if (dds_rc != DDS_RETCODE_OK)
    {
        printf("Failed to get a loan\n");
        return -1;
    }

    /* After this function returns with DDS_RETCODE_OK,
```

(continues on next page)

(continued from previous page)

```
     * the middleware owns the sample
     */
    dds_rc = FooDataWriter_write(hw_datawriter, sample, &DDS_HANDLE_NIL);
}
```

### Reading samples

The following code illustrates how to read samples annotated with `@transfer_mode(SHMEM_REF)`:

```
DDS_ReturnCode_t dds_rc;
dds_rc = FooDataReader_take(...)

/* process sample here */
/* NEW API
   IMPORTANT - is_data_consistent will always return true when ZC v2 is being used
 */

dds_rc = FooDataReader_is_data_consistent(hw_reader,
                                          &is_data_consistent,
                                          sample,sample_info);

if (dds_rc == DDS_RETCODE_OK)
{
     if (is_data_consistent)
     {
         /* Sample is consistent. Processing of sample is valid */
     }
     else
     {
        /* Sample is NOT consistent. Any processing of the sample should
         * be discarded and considered invalid.
         */
     }
}
```

### 3.15.4 Synchronizing samples

Zero Copy v1 and v2 handle sample synchronization differently. The following sections explain these differences in detail.

#### Zero Copy v1 synchronization

In Zero Copy v1, no synchronization exists between the sender (*DataWriter*) and the receiver (*DataReader*) for Zero Copy samples. This means that a sample's content can be invalidated before the receiver has a chance to read it.

For example, consider creating a best-effort *DataWriter* with `max_samples = 1`. When the *DataWriter* is initialized, the middleware pre-allocates a pool of `max_samples` + 1 (2) samples in a shared memory region. These samples are loaned to the *DataWriter* when calling FooDataWriter_get_loan().

The following code illustrates this:

```
DDS_ReturnCode_t ddsrc;
Foo* sample;

ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 1 */
sample->value = 10000;
ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);
/*
* As this is a best-effort writer, the middleware immediately makes
* this sample available for reuse by another FooDataWriter_get_loan(...) call.
*/

ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 2 */
sample->value = 20000;
ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);
/*
* Again, the sample is made available immediately for reuse.
*/

/*
* At this point, the sample may have been received by the DataReader
* but not yet presented to the user.
*/

ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 1 */
/*
* sample->value will now contain 10000 because the sample is reused
* from a pool with only 2 buffers.
*
* Additionally, references to both sample 1 and sample 2 might already
* have been received by the middleware on the *DataReader* side and stored
* in its internal cache. However, these samples may not yet have been delivered
* to the application. If sample->value is modified to 999 at this point,
* a subsequent call to *read()* or *take()* from the Subscriber will return 999,
```

(continues on next page)

(continued from previous page)

```
* not the expected 10000. This happens because both the Publisher and Subscriber
* share the same memory region.
*
* Use `FooDataReader_is_data_consistent` to verify sample consistency and avoid
* this issue.
*
* Note: A sample becomes invalidated right after `FooDataWriter_get_loan(dw, &sample)`
* is completed. If the sample address has already been written to and has not yet
* been read by the receiver, the previously written data is invalidated.
*/

ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);
```

### Zero Copy v2 synchronization

Zero Copy v2 provides synchronization between the *DataWriter* and *DataReader*. Since the queue size is limited, samples are reused once the queue is full. However, if a *DataWriter* modifies a sample before the *DataReader* has accessed it, that sample will not be presented to the user. Additionally, samples currently being read by the *DataReader* are locked, preventing the *DataWriter* from accessing them.

Consider the following example with `max_samples = 1` (internally, the middleware will allocate 2 samples):

```
ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 1 */
sample->value = 10000;
ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);

ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 2 */
sample->value = 20000;
ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);

/* Both samples are now available to the user, but the Reader may not have accessed them␣
↪yet. */

ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 1 */
sample->value = 30000;
ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);
```

If the *DataReader* takes all the samples:

```
FooDataReader_take(reader,
                   &sample_seq, &info_seq,
                   DDS_LENGTH_UNLIMITED,
                   DDS_ANY_SAMPLE_STATE,
                   DDS_ANY_VIEW_STATE,
                   DDS_ANY_INSTANCE_STATE);
```

It will only receive two samples with values of 20000 and 30000, respectively.

If both the samples are currently being accessed by the user by calling FooDataReader_read() or FooDataReader_take(), they will be locked. Any attempt to call FooDataWriter_get_loan() on the *DataWriter* will result in an `OUT_OF_RESOURCES` error.

### 3.15.5 Caveats

- After you call FooDataWriter_write(), the middleware takes ownership of the sample. It is no longer safe to make any changes to the sample that was written. If, for whatever reason, you call FooDataWriter_get_loan() but never write the sample, you must call FooDataWriter_discard_loan() to return the sample back to the FooDataWriter. Otherwise, subsequently calling FooDataWriter_get_loan() may fail, because the FooDataWriter has no samples to loan.
- The current maximum supported sample size is a little under the maximum value of a signed 32-bit integer. For that reason, do not use any samples greater than 2000000000 bytes.

## 3.16 FlatData Language Binding

This section is organized as follows:

- *Overview*
- *Getting Started*
- *Further Information*

### 3.16.1 Overview

*Connext Micro* supports the FlatData™ language binding in the same manner as *Connext Professional*. However, *Connext Micro* only supports the FlatData language binding for traditional C++ APIs, whereas *Connext Professional* also supports it for the Modern C++ API. The FlatData language binding is not supported for the C language binding.

### 3.16.2 Getting Started

The best way to start is to generate an example by creating an example IDL file HelloWorld.idl containing the following IDL type:

```
@final
@language_binding(FLAT_DATA)
struct HelloWorld
{
   long a;
}
```

Next, run:

```
rtiddsgen -example -micro -language C++ HelloWorld.idl
```

### 3.16.3 Further Information

For more details about this feature, please see the FlatData Language Binding section in the RTI Connext DDS Core Libraries User's Manual.

For details on how to build and read a FlatData sample, see FlatData.

## 3.17 Application Generation Using XML

*RTI Connext Micro*'s Application Generation feature enables you to specify an application in XML. It simplifies and accelerates application development by enabling the creation of DDS *Entities* (and registration of the factories) used in an application by compiling an XML configuration file, linking the result to an application, and calling a single API. Once created, all *Entities* can be retrieved from the application code using standard "lookup_by_name" operations so that they can be used to read and write data. UDP transport, DPDE (Dynamic Participant Dynamic Endpoint), and DPSE (Dynamic Participant Static Endpoint) discovery configuration can also be configured as needed. C or C++ source code is generated from the XML configuration and compiled with the application.

Once you have your XML file definition, you must use the Micro Application Generator (MAG) tool to load the XML file definition into *Connext Micro*. MAG is needed because *Connext Micro* does not include an XML parser (this would significantly increase code size and amount of memory needed). MAG generates C source code from the XML configuration that you must then compile with the application. The generated C source code contains the same information as the XML configuration file. The generated C source code can be used from both the C API Reference and C++ API Reference.

The *Connext Micro* Application Generation is enabled by default in this release when compiling with rtime-make. However, future releases may disable the feature by default. Thus, it is advised to always compile with the *Connext Micro* Application Generation feature enabled (-DRTIME_DDS_ENABLE_APPGEN=1 to CMake).

**Warning:** Micro Application Generation is only tested with UTF-8 encoding. Other encoding standards are untested and can result in unexpected behavior.

### 3.17.1 Defining an Application in XML

Each *Entity* configured in the XML file is given a name. This name is used to retrieve the entities at runtime using the *Connext Micro* API.

In the XML file, you need to distinguish between two names:

- Configuration name: The name of a specific *Entity*'s configuration. It is given by the name attribute of the corresponding element.
- Entity name in the *Entity*'s QoS: The Entity name in the *Entity*'s QoS.

At runtime, the *Entity* will be created using the Entity name in the *Entity*'s QoS; the configuration name will be used if this is an empty string.

The attribute multiplicity indicates that a set of *Entities* should be created from the same configuration. Since each *Entity* must have a unique name, the system will automatically append a number to the Entity name in the *Entity*'s QoS (or, if it is an empty string, the configuration name) to obtain the Entity name. For example, if we specified a multiplicity of "N", then for each index "i" between 0 and N-1, the system will assign Entity names according to the table below:

| Entity Name | Index: i |
|---|---|
| "configuration_name" | 0 |
| "configuration_name#i" | [1,N-1] |

That is, the *Entity* name followed by the token "#" and an index.

See *Generate Type-Support Code from the Type Definition* for an example XML file.

#### Important Points

Applications can create a *RTI Connext Micro Entity* from a *DomainParticipant* configuration described in the XML configuration file. All the *Entities* defined by such *DomainParticipant* configuration are created automatically as part of the creation. In addition, multiple *DomainParticipant* configurations may be defined within a single XML configuration file.

All the *Entities* created from a *DomainParticipant* configuration are automatically assigned an entity name. *Entities* can be retrieved via "lookup_by_name" operations specifying their name. Each *Entity* stores its own name in the QoS policies of the *Entity* so that they can be retrieved locally (via a lookup) and communicated via discovery.

A configuration file is not tied to the application that uses it. Different applications may run using the same configuration file. A single file may define multiple *DomainParticipant* configurations. Normally, a single application will instantiate only one *DomainParticipant*, but an application can instantiate as many *DomainParticipants* as needed.

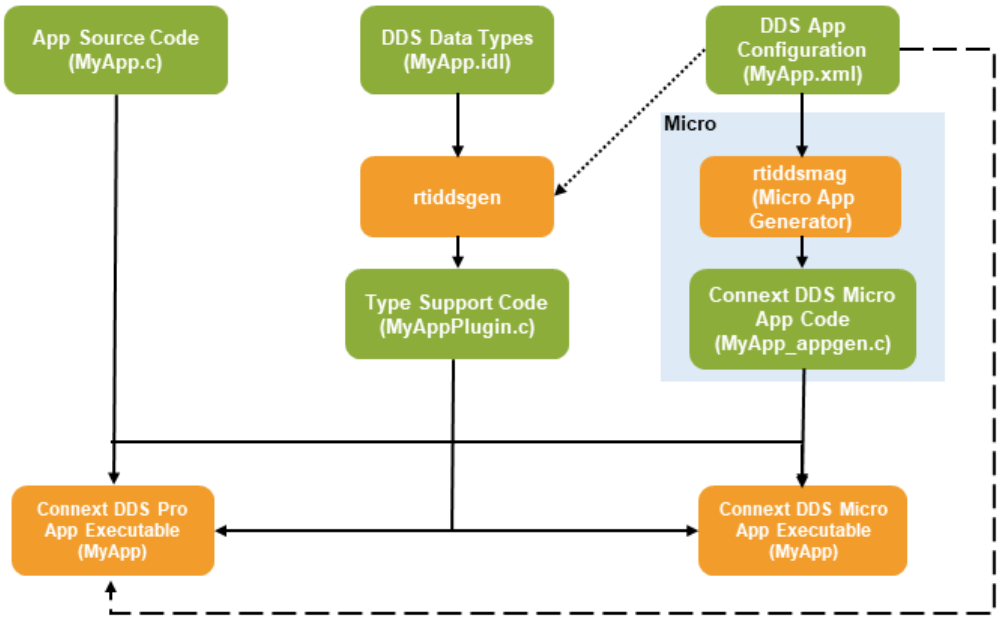
Changes in the XML configuration file require regenerating the C/C++ source code and recompiling the application.

### 3.17.2 Generating the Application from XML

*Connext Micro* comes with a tool, the Micro Application Generator (MAG). This tool is used to generate supporting files to create XML-defined applications at runtime.

#### Micro Application Generator (MAG) Tool

Micro Application Generator (MAG) is a required tool to configure *Connext Micro* applications by generating code from an XML configuration file; it creates DDS entities and registers all the components needed for a *Connext Micro*-based application. MAG can process your own XML configuration file, or it can process an *XML-Based Application Creation* file that you created for *RTI Connext Professional*.




*Connext Micro* Application Generation, in combination with MAG, enables two important use cases:

- Users who may eventually develop with *Connext Micro*, but who haven't determined their final platform, can prototype applications on a generic platform and validate that the QoS and DDS Entity configuration is within scope of what *Connext Micro* supports. MAG ignores fields in the XML file that *Connext Micro* doesn't use (and produces an error for the few fields it cannot ignore; see "Unsupported values" in *Errors caused by invalid configurations*).
- Users who want to develop directly with *Connext Micro* can simplify their development efforts through shared XML files that can be configuration-managed. This reduces the burden on system integrators who want to configure *Connext Micro* systems without having to manually code in static configurations.

The main features of MAG are:

- Generates code for the languages supported by *Connext Micro*: C and C++.

- Automatically configures the remote entities that are needed to communicate with applications that use static discovery.
- Automatically tries to use the default values used by *Connext Micro*, to reduce the size of the generated code.
- Optimizes the components used by your application. By default, MAG generates code that will unregister transports that your application is not using.
- Ignores fields and transports not supported by *Connext Micro* (any fields or transports not described in the API Reference) and raises errors for things it can't ignore. See *Errors caused by invalid configurations*.

---

**Note:**

- MAG has been tested with Java 17.0.6, which is included in the *Connext Professional* installation.
- MAG does not support customizable templates. (It doesn't support the functionality described in Customizing the Generated Code in the *Code Generator User's Manual*.)

---

### Generating the Application with MAG

#### Running MAG

To run the MAG tool, use the following command:

For example, on a Windows system:

```
<RTIMEHOME>/rtiddsmag/scripts/rtiddsmag.bat -language C -referencedFile HelloWorldQos.
↪xml HelloWorld.xml
```

For example, on a Linux or macOS system:

```
<RTIMEHOME>/rtiddsmag/scripts/rtiddsmag -language C -referencedFile HelloWorldQos.xml␣
↪HelloWorld.xml
```

Please refer to *MAG Command-Line Options* for valid command-line options.

#### Generated Files

The following table shows the files that MAG creates for an example XML file, **HelloWorld.xml** (which contains the application definition) and a referenced file, **HelloWorldQos.xml** (which contains the QoS definition). This second file is optional; you can define the QoS in the application file.

**Note:** Changes in the XML configuration file require regenerating the C/C++ source code and recompiling the application.

Table 3.11: C and C++ Files Created for Example HelloWorld.xml

| Generated Files | Description |
|---|---|
| HelloWorldAppgen.h (C and C++) | Generated code for each DDS *Entity* and its run-time components. |
| HelloWorldAppgen.c (C and C++) | Generated code for each Entity Model; also contains the values of each array used in the header file. |
| HelloWorldAppgen_plugin.h (C++ only) | Header file that contains the declarations of all the wrappers. |
| HelloWorldAppgen_plugin.cxx (C++ only) | A wrapper for the _get() call (get_plugin_type):<br>`struct DDS_TypePluginI *HelloWorldPlugin_get_cpp(void)`<br>`{`<br>`  return HelloWorldPlugin_get();`<br>`}` |

**Warning:** You should not modify the generated code. MAG will overwrite your modifications when it regenerates the C/C++ code from XML if the `-replace` argument is used.

### MAG Command-Line Options

The following table shows the options available when using *rtiddsmag* to generate code for *Connext Micro* applications.

Table 3.12: Command-Line Options for rtiddsmag

| Option | Description |
|---|---|
| `-applicationType <type>` | Controls the structure and templates used in code generation. Valid values are `micro4`, `cert`, and `AUTOSAR_CDD`. `AUTOSAR_CDD` generates AUTOSAR-compliant code structure, including adapter layer separation and AUTOSAR resource configuration templates. |

continues on next page

Table 3.12 – continued from previous page

| Option | Description |
|---|---|
| `-autosarTemplatePath <template directory>` | Specify the directory where the AUTOSAR templates are.<br>The default path for AUTOSAR code generation templates is `RTIMEHOME/rtiddsmag/resource/templates/autosar`. However, in some cases, it may be necessary to customize the generated code. In such cases, the `-autosarTemplatePath` option can be used to specify the location of customized templates.<br>**Note:** This option is only valid when `-applicationType AUTOSAR_CDD` is specified. The templates must have the same directory structure and names as the ones provided in the default location. |
| `-d <outdir>` | Generates the output in the specified directory. By default, MAG will generate files in the directory where the XML file is located. |
| `-deployment <name>` | Generates code for a single deployment instance, selected from a deployment scenario defined in the XML file. When this option is specified, *rtiddsmag* will:<br>• Parse the selected deployment configuration;<br>• Resolve all application references for that deployment;<br>• Filter participants to only those required by the deployment;<br>• Generate code optimized for the target deployment environment.<br>This capability is essential for multi-deployment scenarios where different Electronic Control Units (ECUs) require different DDS configurations and code artifacts. |
| `-dontAddLocations` | Use this flag to avoid adding the input file location of fields into the generated files.<br>By default (when this flag is not used), MAG will add the location where an entity was defined in the XML file. The location will be placed above the definition of that entity in the generated code. |
| `-dontOptimizeSE` | Use this flag to avoid static endpoint discovery optimization. Then MAG will include all *DataWriters* and *DataReaders* when calculating the remote entities.<br>By default (when this option is not used) MAG will optimize the number of remote entities by only including Data Writers and *DataReaders* that use the same Topic in the remote model. |

continues on next page

Table 3.12 – continued from previous page

| Option | Description |
|---|---|
| `-dontUpdateResourceLimits` | Use this flag to avoid automatically updating the resource limit settings for the DomainParticipantFactory, *DomainParticipants*, *DataReaders*, and *DataWriters*.<br>**Note:** The use of this flag for the DomainParticipantFactory is currently not supported.<br>By default (when this flag is not used), MAG will update the resource limits so it will at least be able to support the entities defined in the XML file. If your applications communicate with more remote entities that the ones specified in the XML file, you might need to manually update them. |
| `-dontUseDefaultValues` | Use this flag to avoid automatically generating code using default QoS policy values when possible.<br>By default (when this flag is not used), MAG will check whether the values that are set in every element of the QoS policies for each entity are the same as the defaults used by *Connext Micro*. If that's the case, the generated code will contain the default values for those policies, instead of the values set by the user. |
| `-dpdeName <name>` | Specifies the name used by MAG when registering a DPDE discovery plugin. By default, this name is `dpde`. *rtiddsmag* will also append a *DomainParticipant* number to the default name for each unique configuration. |
| `-dpseName <name>` | Specifies the name used by MAG when registering a DPSE discovery plugin. By default, this name is `dpse`. *rtiddsmag* will also append a *DomainParticipant* number to the default name for each unique configuration. |
| `-help` | Prints out the command-line options for MAG. |
| `-idlFile <file>` | Specifies the IDL file name used by rtiddsgen to generate the code. This value is used by MAG to specify the Plugin header generated by rtiddsgen. By default, MAG uses the name of the XML file. |
| `-inputXml <file>` | Specifies the XML configuration file used to generate code. The XML configuration file can be passed directly to MAG without using the -inputXml option, by default MAG knows that any argument with no option is the input file. |
| `-language <C\|C++>` | Specifies the language to use for the generated files. The default language is C. |
| `-onlyValidate` | Causes MAG to just validate the input file. It will not generate any code. |

continues on next page

Table 3.12 – continued from previous page

| Option | Description |
|---|---|
| `-outputFinalQoS <QosLibrary::QosProfile>` | Use this flag to display the final values of the specified QoS profile after applying inheritance.<br>Although MAG currently doesn't generate code to set the QoS for *Connext Micro*, using this flag will determine the final values in the profile after applying inheritance. For complex XML files, with multiple levels of inheritance, it might be a challenge to determine the final QoS values. Using this flag simplifies the process.<br>**Note:** This option does not check whether or not the final values are supported by *Connext Micro*. |
| `-pskName <name>` | Change the name of the pre-shared key (PSK) component for the *Lightweight Security Plugin*. The default name is `psk`. *rtiddsmag* will also append a *DomainParticipant* number to the default name for each unique configuration. |
| `-referencedFile <file>` | Specifies a file which is referenced from the one being used to generate code.<br>In general, it is recommended to split the application definition from the QoS definition. This way, the QoS can be shared among various applications.<br>**Note:** Can be repeated. `-referencedFile <file1> -referencedFile <file2> ...` |
| `-replace` | Use this flag to overwrite existing generated files. |
| `-shmemName <name>` | Specifies the name used by MAG when registering a shared memory (SHMEM) transport plugin. By default, this name is `shmem`. *rtiddsmag* will also append a *DomainParticipant* number to the default name for each unique configuration. |
| `-udpName <name>` | Specifies the name used by MAG when registering a UDP transport plugin. By default, this name is `udpv4`. |
| `-incrementUdpName` | Use this flag to add the `participant_number` suffix to the UDP transport name. By default, the UDP transport name does not have the participant_number suffix. |
| `-verbosity [1-4]` | Sets the MAG verbosity:<br>1: Exceptions.<br>2: Exceptions and warnings. **(Default)**<br>3: Exceptions, warnings, and information.<br>4: Exceptions, warnings, information, and debug. |
| `-version` | Displays the version of MAG being used, such as 4.3.0. |
| `-zCopyName <zcopy name>` | Specifies the name MAG uses when registering a Zero Copy v2 transport plugin. By default, this name is `zcopy`. *rtiddsmag* will also append a *DomainParticipant* number to the default name for each unique configuration. |

#### Integrating Generated Files into Your Application's Build

Integrating the generated files into your application is as easy as including the generated files **HelloWorldAppgen.h** and **HelloWorldAppgen.c** in your application. If your application uses C++, you will also need to include **HelloWorldAppgen_plugin.h** and **HelloWorldAppgen_plugin.cxx**.

Then you can create entities using the standard `DDS_DomainParticipantFactory_create_participant_from_cor` operation and retrieve all the entities from your application code using the standard `lookup_<entity>_by_name()` operations, such as `lookup_datawriter_by_name()`. For details on these operations, see the DomainParticipantFactory module in the *Connext Micro* API reference HTML documentation.

### 3.17.3 Creating the Application

We'll create an example application using *rtiddsgen* as seen in *Data Types* and learn from the generated files.

#### Generate Type-Support Code from the Type Definition

The first step is to describe the data type in a programming language-neutral manner. Three languages are supported by *RTI Code Generator*: XML, IDL, and XSD. These three languages provide equivalent type-definition capabilities, so you can choose whichever one you prefer. You can even transform between one of these three languages and another with *RTI Code Generator*. That said, since the rest of the configuration files use XML, it is often more convenient to also use XML to describe the data types, so they can be shared or moved to other XML configuration files.

Create a file called **HelloWorld.xml** that contains the following XML content:

```
<types>
    <const name="MAX_NAME_LEN" type="long" value="64"/>
    <const name="MAX_MSG_LEN"  type="long" value="128"/>
    <struct name="HelloWorld">
        <member name="sender"  type="string" stringMaxLength="MAX_NAME_LEN" key="true"/>
        <member name="message" type="string" stringMaxLength="MAX_MSG_LEN"/>
        <member name="count"   type="long"/>
    </struct>
</types>
```

We'll use this data type for the following example. The data associated with the HelloWorld *Topic* consists of two strings and a numeric counter:

1. The first string contains the name of the sender of the message. This field is marked as the "key" since it signals the identity of the data-object.
2. The second string contains a message.
3. The third field is a simple counter, which the application increments with each message.

Once the type has been defined, we use *rtiddsgen* to generate the code for the HelloWorld data type.

We will generate a DPDE example. From your command shell, change to the directory where you created **HelloWorld.xml** and type:

**To generate code with rtiddsgen:**

- On a Windows system:

```
<RTIMEHOME>\rtiddsgen\scripts\rtiddsgen.bat -example -exampleTemplate mag/dpde -
↪language C HelloWorld.xml -replace
```

- On a Linux or macOS system:

```
<RTIMEHOME>/rtiddsgen/scripts/rtiddsgen -example -exampleTemplate mag/dpde -
↪language C HelloWorld.xml -replace
```

After running *rtiddsgen*, you will see **HelloWorldQos.xml** and the following files and their associated header files in the `HelloWorld_mag_dpde` directory:

- **HelloWorld.c**
- **HelloWorldPlugin.c**
- **HelloWorldSupport.c**
- **HelloWorldAppgen.c**
- **HelloWorld_publisher.c**
- **HelloWorld_subscriber.c**
- **HelloWorldApplication.c**

The most notable files are **HelloWorld.h** and **HelloWorldPlugin.h**:

- **HelloWorld.h** contains the declaration of the C structure, built according to the specification in the XML file:

```
#define MAX_NAME_LEN (64L)
#define MAX_MSG_LEN (128L)
typedef struct HelloWorld
{
    DDS_String   sender;
    DDS_String   message;
    DDS_Long   count;
} HelloWorld;
```

- **HelloWorldPlugin.h** contains the `TypePlugin_get()` function that MAG will use when generating the code to create all the DDS entities:

```
NDDSUSERDllExport extern struct NDDS_Type_Plugin*
    HelloWorldTypePlugin_get(void);
```

### Generate DDS Entities from the System Definition

This step uses *rtiddsmag* to generate code to support the creation of DDS entities using Application Generation in *Connext Micro*.

*rtiddsmag* supports C and C++. We will generate the DPDE example.

---

**Note:** Type code doesn't need to exist when running *rtiddsmag*. However, we'll be using the HelloWorldQos.xml and HelloWorld.xml files generated from the *Generate Type-Support Code from the Type Definition* section.

---

From your command shell, change to the directory where you created **HelloWorld.xml** and type:

**To generate code with rtiddsmag:**

- On a Windows system:

```
<RTIMEHOME>\rtiddsmag\scripts\rtiddsmag.bat -language C -referencedFile␣
↪HelloWorldQos.xml HelloWorld.xml
```

- On a Linux or macOS system:

```
<RTIMEHOME>/rtiddsmag/scripts/rtiddsmag -language C -referencedFile HelloWorldQos.
↪xml HelloWorld.xml
```

We will examine the content of the generated files in the next section.

### Examining the application

*rtiddsmag* generates two applications: **HelloWorld_publisher**, which writes the *Topic*, HelloWorldTopic, and **HelloWorld_subscriber**, which subscribes to that *Topic*.

*rtiddsmag* also generates the following examples from the DPSE and the DPDE directories:

- **Domain Participant "HelloWorldDPDEPubDP"**

  This application defines a publisher which uses DPDE discovery.

  The application has one named "HelloWorldDPDEPubDP", one named "HelloWorldDPDEPub", and one named "HelloWorldDPDEDW" which uses topic name "Example HelloWorld". The application registers one type with name "HelloWorld" and defines one with name "Example HelloWorld" which uses the type "HelloWorld".

- **Domain Participant "HelloWorldDPDESubDP"**

  This application defines a subscriber which uses DPDE discovery.

  The application has one named "HelloWorldDPDESubDP", one named "HelloWorldDPDESub", and one named "HelloWorldDPDEDR" which uses topic name "Example HelloWorld". The application registers one type with name "HelloWorld" and defines one with name "Example HelloWorld" which uses the type "HelloWorld".

#### Call API to Create DomainParticipant

Call the API create_participant_from_config() to create the application from the generated XML definition. This API creates the *DomainParticipant*; all applications start with the *DomainParticipant*. This API receives the configuration name and creates all the *Entities* defined by that configuration.

#### Retrieve Entities by Name

After *DomainParticipant* creation, you can retrieve the defined *Entities* by using lookup_by_name() operations.

### 3.17.4 Examine example XML configuration files

An entire **HelloWorld.xml** file is shown below. Let's review its content to see how this scenario was constructed. The main sections in the file are:

- *Type Definition*
- *Domain Definition*
- *DomainParticipant Definition*

```
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/current/rti_dds_profiles.
↪xsd">
    <!-- Type Definition -->
    <types>
        <const name="MAX_NAME_LEN" type="int32" value="64"/>
        <const name="MAX_MSG_LEN"  type="int32" value="128"/>
        <struct name="HelloWorld">
            <member name="sender"  type="string" stringMaxLength="MAX_NAME_LEN" key="true
↪"/>
            <member name="message" type="string" stringMaxLength="MAX_MSG_LEN"/>
            <member name="count"   type="int32"/>
        </struct>
    </types>
    <!-- Domain Library -->
    <domain_library name="HelloWorldLibrary">
        <domain name="HelloWorldDomain" domain_id="0">
            <register_type name="HelloWorldType" type_ref="HelloWorld">
            </register_type>
            <topic name="HelloWorldTopic" register_type_ref="HelloWorldType">
                <registered_name>HelloWorldTopic</registered_name>
            </topic>
        </domain>
    </domain_library>
    <!-- Participant Library -->
    <domain_participant_library name="HelloWorldAppLibrary">
```

(continues on next page)

(continued from previous page)

```
        <domain_participant name="HelloWorldDPDEPubDP"
         domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <publisher name="HelloWorldDPDEPub">
                <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPDEDW">
                    <datawriter_qos base_name="QosLibrary::DPDEProfile"/>
                </data_writer>
            </publisher>
            <domain_participant_qos base_name="QosLibrary::DPDEProfile"/>
        </domain_participant>
        <domain_participant name="HelloWorldDPDESubDP"
         domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <subscriber name="HelloWorldDPDESub">
                <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPDEDR">
                    <datareader_qos base_name="QosLibrary::DPDEProfile"/>
                </data_reader>
            </subscriber>
            <domain_participant_qos base_name="QosLibrary::DPDEProfile"/>
        </domain_participant>
        <domain_participant name="HelloWorldDPSEPubDP"
         domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <publisher name="HelloWorldDPSEPub">
                <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPSEDW">
                <datawriter_qos base_name="QosLibrary::DPSEProfile"/>
                </data_writer>
            </publisher>
            <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
        </domain_participant>
        <domain_participant name="HelloWorldDPSESubDP"
         domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <subscriber name="HelloWorldDPSESub">
                <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPSEDR">
                    <datareader_qos base_name="QosLibrary::DPSEProfile"/>
                </data_reader>
            </subscriber>
            <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
        </domain_participant>
    </domain_participant_library>
</dds>
```

### Type Definition

*rtiddsmag* doesn't use the types section of the XML file to generate any code. This section is used by *rtiddsgen* to generate the code to support the direct use of the structure 'HelloWorld' from application code (see *Generate Type-Support Code from the Type Definition*).

```
<types>
    <const name="MAX_NAME_LEN" type="int32" value="64"/>
    <const name="MAX_MSG_LEN"  type="int32" value="128"/>
    <struct name="HelloWorld">
        <member name="sender"  type="string" stringMaxLength="MAX_NAME_LEN" key="true"/>
```

(continues on next page)

(continued from previous page)

```
        <member name="message" type="string" stringMaxLength="MAX_MSG_LEN"/>
        <member name="count"   type="int32"/>
    </struct>
</types>
```

### Domain Definition

The domain section defines the system's *Topics* and their corresponding data types. To define a *Topic*, the associated data type must be registered with the domain, giving it a registered type name. The registered type name is used to refer to that data type within the domain when the *Topic* is defined.

In this example, the configuration file registers the previously defined HelloWorld type under the name HelloWorldType. Then it defines a *Topic* named HelloWorldTopic, which is associated with the registered type, referring to its registered name, HelloWorldType. The value used in `TypePlugin_get` depends on how the registration of the data-type is configured inside the domain:

1. If a <register_type> tag is specified *without* a type_ref attribute, the value of `TypePlugin_get` is generated from the <register_type> tag plus the string "Plugin_get".
2. If a <register_type> tag is specified *with* a type_ref attribute, the value of `TypePlugin_get` is generated from that attribute plus the string "TypePlugin_get". Our example has type_ref = "HelloWorld", so the value of `TypePlugin_get` will be `HelloWorldTypePlugin_get`.

```
<!-- Domain Library -->
<domain_library name="HelloWorldLibrary">
    <domain name="HelloWorldDomain" domain_id="0">
        <register_type name="HelloWorld" type_ref="HelloWorld">
        </register_type>

        <topic name="HelloWorldTopic" register_type_ref="HelloWorld">
            <registered_name>HelloWorldTopic</registered_name>
        </topic>
    </domain>
</domain_library>
```

*rtiddsmag* generates the following code for each entity that uses this *Topic*:

- **HelloWorldAppgen.c**

```
const struct APPGEN_TypeRegistrationModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_type_registrations[1] =
{
   {
      "HelloWorldType", /* registered_type_name */
      HelloWorldTypePlugin_get /* get_type_plugin */
   }
};
const struct APPGEN_TopicModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_topics[1] =
```

(continues on next page)

(continued from previous page)

```
{
   {
      "HelloWorldTopic", /* topic_name */
      "HelloWorldType", /* type_name */
      DDS_TopicQos_INITIALIZER /* topic_qos*/
   }
};
```

These two structures are used in the *DomainParticipant* definition, where they will be registered by *Connext Micro* when calling the Micro Application Generation API.

- **HelloWorldAppgen.h**

```
extern const struct APPGEN_TypeRegistrationModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_type_registrations[1];

extern const struct APPGEN_TopicModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_topics[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
   ...
   1UL, /* type_registration_count */ \
   HelloWorldAppLibrary_HelloWorldDPDEPubDP_type_registrations, /* type_
↪registrations*/ \
   1UL, /* topic_count */ \
   HelloWorldAppLibrary_HelloWorldDPDEPubDP_topics, /* topics */ \
   ...
}
```

**Note:** *Connext Micro* automatically registers the types that *rtiddsmag* generates. This means the content inside the Domain definition must match the types generated by *rtiddsgen*.

### DomainParticipant Definition

The *DomainParticipant* section defines the *DomainParticipants* in the system and the *DataWriters* and *DataReaders* that each *DomainParticipant* has. *DomainParticipants* are defined within the <domain_participant_library> tag.

Each *DomainParticipant*:

- Has a unique name (within the library) which will be used later by the application that creates it.
- Is associated with a domain, which defines the `domain_id`, *Topics*, and the data types the *DomainParticipant* will use.
- Defines the *Publishers* and *Subscribers* within the *DomainParticipant*. *Publishers* contain *DataWriters*, *Subscribers* contain *DataReaders*.

- Defines the set of *DataReaders* it will use to read data. Each *DataReader* has a QoS and a unique name which can be used from application code to retrieve it.
- Defines the set of *DataWriters* it will use to write data. Each *DataWriter* has a QoS and a unique name which can be used from application code to retrieve it.
- Optionally, the *DomainParticipants*, *Publishers*, *Subscribers*, *DataWriters*, and *DataReaders* can specify a QoS profile that will be used to configure them.

The example below defines four *DomainParticipants*, two of them (HelloWorldDPDEPubDP and HelloWorldDPDESubDP) use Dynamic Participant/Dynamic Endpoint (DPDE) and the other two (HelloWorldDPSEPubDP and HelloWorldDPSESubDP) use Dynamic Participant/Static Endpoint (DPSE) discovery:

```
<!-- Participant Library -->
<domain_participant_library name="HelloWorldAppLibrary">
    <domain_participant name="HelloWorldDPDEPubDP"
     domain_ref="HelloWorldLibrary::HelloWorldDomain">
        <publisher name="HelloWorldDPDEPub">
            <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPDEDW">
                <datawriter_qos base_name="QosLibrary::DPDEProfile"/>
            </data_writer>
        </publisher>
        <domain_participant_qos base_name="QosLibrary::DPDEProfile"/>
    </domain_participant>

    <domain_participant name="HelloWorldDPDESubDP"
     domain_ref="HelloWorldLibrary::HelloWorldDomain">
        <subscriber name="HelloWorldDPDESub">
            <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPDEDR">
                <datareader_qos base_name="QosLibrary::DPDEProfile"/>
            </data_reader>
        </subscriber>
        <domain_participant_qos base_name="QosLibrary::DPDEProfile"/>
    </domain_participant>

    <domain_participant name="HelloWorldDPSEPubDP"
     domain_ref="HelloWorldLibrary::HelloWorldDomain">
        <publisher name="HelloWorldDPSEPub">
            <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPSEDW">
                <datawriter_qos base_name="QosLibrary::DPSEProfile"/>
            </data_writer>
        </publisher>
        <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
    </domain_participant>

    <domain_participant name="HelloWorldDPSESubDP"
     domain_ref="HelloWorldLibrary::HelloWorldDomain">
        <subscriber name="HelloWorldDPSESub">
            <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPSEDR">
                <datareader_qos base_name="QosLibrary::DPSEProfile"/>
            </data_reader>
        </subscriber>
```

(continues on next page)

(continued from previous page)

```
        <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
    </domain_participant>
</domain_participant_library>
```

Examining the XML, we see that:

- Each *DomainParticipant* is bound to the Domain, HelloWorldLibrary::HelloWorldDomain.
- The two *DomainParticipants* that use DPDE as their discovery mechanism inherit from the profile QosLibrary::DPDELibrary, while the other two that use DPSE as their discovery mechanism inherit from QosLibrary::DPSELibrary.
- Each *DomainParticipant* contains a single *Publisher* or *Subscriber*, which it turn contains a single *DataWriter* or *DataReader* that inherits from QosLibrary::DPDELibrary or QosLibrary::DPSELibrary, depending on the discovery mechanism used by its *DomainParticipant*.
- Each *DataWriter* writes the *Topic* HelloWorldTopic, which is defined in the domain HelloWorldLibrary::HelloWorldDomain. Each *DataReader* reads the same *Topic*.

Since both Dynamic *DomainParticipants* (those which are using DPDE as their discovery mechanism) are in the same the domain and the *DataWriter* writes the same *Topic* that the *DataReader* reads, the two *DomainParticipants* will communicate. This also apply to both static participants (those which are using DPSE as their discovery mechanism); the only difference is that *rtiddsmag* will generate extra code to configure the remote entities (for details, see *Static Discovery*).

Let's look at the content of a *DomainParticipant* definition to explain the code generated by *rtiddsmag*.

```
<domain_participant name="HelloWorldDPDEPubDP"
 domain_ref="HelloWorldLibrary::HelloWorldDomain">
    <publisher name="HelloWorldDPDEPub">
        <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPDEDW">
            <datawriter_qos base_name="QosLibrary::DPDEProfile"/>
        </data_writer>
    </publisher>
    <domain_participant_qos base_name="QosLibrary::DPDEProfile"/>
</domain_participant>
```

*rtiddsmag* generates the code needed to register each component used by this *DomainParticipant* and unregister those components that are not being used. In our example, for each *DomainParticipant*, *rtiddsmag* registers the discovery transport, `dpde` or `dpse`; registers the UDP transport used by each *DomainParticipant* (since they use the same configuration, only one UDP transport configuration is generated); and unregisters the default UDP and INTRA transports, since they are not being used (these two are the only ones that can be unregistered by *rtiddsmag*).

It also creates the code for each entity. In this case, it generates the code needed to create:

- A *Publisher* named HelloWorldDPDEPub
- A *DataWriter* named HelloWorldDPDEDW
- A *DomainParticipant* named HelloWorldDPDEPubDP

- The QoS used by this *DomainParticipant* (see *QoS Definition*)

**HelloWorldAppgen.c**

```
const struct ComponentFactoryUnregisterModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_unregister_components[2] =
{
    {
        "_udp", /* NETIO_DEFAULT_UDP_NAME */
        NULL, /* udp struct RT_ComponentFactoryProperty** */
        NULL  /* udp struct RT_ComponentFactoryListener** */
    },
    {
        "_intra", /* NETIO_DEFAULT_INTRA_NAME */
        NULL, /* _intra struct RT_ComponentFactoryProperty** */
        NULL  /* _intra struct RT_ComponentFactoryListener** */
    }
};

const struct DPDE_DiscoveryPluginProperty
HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde[1] =
{
    RTI_APP_GEN___dpde__HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde1
};
const struct UDP_InterfaceFactoryProperty
HelloWorldAppLibrary_HelloWorldDPDEPubDP_udpv4[1] =
{
    RTI_APP_GEN___udpv4__HelloWorldAppLibrary_HelloWorldDPDEPubDP_udp1
};
const struct ComponentFactoryRegisterModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_register_components[2] =
{
    {
        "dpde1", /* register_name */
        DPDE_DiscoveryFactory_get_interface, /* register_intf */
        &HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde[0]._parent, /* register_property␣
↪*/
        NULL /* register_listener */
    },
    {
        "udp1", /* register_name */
        UDP_InterfaceFactory_get_interface, /* register_intf */
        &HelloWorldAppLibrary_HelloWorldDPDEPubDP_udpv4[0]._parent._parent, /* register_
↪property */
        NULL /* register_listener */
    }
};

...

const struct APPGEN_DataWriterModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_publisher_HelloWorldDPDEPub_data_writers[1] =
{
```

(continues on next page)

(continued from previous page)

```
    {
        "HelloWorldDPDEDW", /* name */
        1UL, /* multiplicity */
        "HelloWorldTopic", /* topic_name */
        RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_
↪HelloWorldDPDEDW /* writer_qos */
    }
};
const struct APPGEN_PublisherModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_publishers[1] =
{
    {
        "HelloWorldDPDEPub", /* name */
        1UL, /* multiplicity */
        DDS_PublisherQos_INITIALIZER, /* publisher_qos */
        1UL, /* writer_count */
        HelloWorldAppLibrary_HelloWorldDPDEPubDP_publisher_HelloWorldDPDEPub_data_
↪writers /* data_writers */
    }
};
```

### HelloWorldAppgen.h

```
extern const struct DPDE_DiscoveryPluginProperty HelloWorldAppLibrary_
↪HelloWorldDPDEPubDP_dpde[1];
extern const struct UDP_InterfaceFactoryProperty HelloWorldAppLibrary_
↪HelloWorldDPDEPubDP_udpv4[1];
extern const struct ComponentFactoryUnregisterModel
  HelloWorldAppLibrary_HelloWorldDPDEPubDP_unregister_components[2];
extern const struct ComponentFactoryRegisterModel
  HelloWorldAppLibrary_HelloWorldDPDEPubDP_register_components[2];

#define RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    2UL, /* unregister_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_unregister_components, /* unregister_
↪components */\
    2UL, /* register_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_register_components, /* register_components␣
↪*/ \
    RTI_APP_GEN___DPF_QOS_QosLibrary_DefaultProfile /* factory_qos */ \
}

extern const struct APPGEN_TypeRegistrationModel
  HelloWorldAppLibrary_HelloWorldDPDEPubDP_type_registrations[1];
extern const struct APPGEN_TopicModel HelloWorldAppLibrary_HelloWorldDPDEPubDP_topics[1];
extern const struct APPGEN_PublisherModel
  HelloWorldAppLibrary_HelloWorldDPDEPubDP_publishers[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
```

(continues on next page)

(continued from previous page)

```
    "HelloWorldDPDEPubDP", /* name */ \
    RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPDEPubDP, /* domain_participant_
↪factory */ \
    RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP, /* domain_participant_
↪qos */ \
    0L, /* domain_id */ \
    1UL, /* type_registration_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_type_registrations, /* type_registrations */
↪ \
    1UL, /* topic_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_topics, /* topics */ \
    1UL, /* publisher_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_publishers, /* publishers */ \
    0UL, /* subscriber_count */ \
    NULL, /* subscribers */ \
    0UL, /* remote_participant_count */ \
    NULL, /* remote_participants */ \
    0UL, /* custom_flow_controller_count */ \
    NULL /* custom_flow_controllers */ \
}
```

### QoS Definition

The defined DDS Entities have an associated QoS Policy, which can be defined in a separate file such as **HelloWorldQos.xml** or within the System XML file.

For more information on how to configure DDS Entities in an XML file, see Configuring QoS with XML (if you have internet access).

See the entire file below. Then we will examine the file section by section, showing the code generated by *rtiddsmag* for the DPSE example.

```
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/current/rti_dds_profiles.
↪xsd">
    <qos_library name="QosLibrary">
        <qos_profile name="DefaultProfile" is_default_participant_factory_profile="true">

            <!-- Participant Factory Qos -->
            <participant_factory_qos>
                <entity_factory>
                    <autoenable_created_entities>false</autoenable_created_entities>
                </entity_factory>
            </participant_factory_qos>

            <!-- Participant Qos -->
            <domain_participant_qos>
                <discovery>
                    <accept_unknown_peers>false</accept_unknown_peers>
```

(continues on next page)

(continued from previous page)

```
                <initial_peers>
                    <element>127.0.0.1</element>
                    <element>239.255.0.1</element>
                </initial_peers>
                <enabled_transports>
                    <element>udpv4</element>
                </enabled_transports>
                <multicast_receive_addresses>
                    <element>udpv4://127.0.0.1</element>
                    <element>udpv4://239.255.0.1</element>
                </multicast_receive_addresses>
            </discovery>
            <default_unicast>
                <value>
                    <element>
                        <transports>
                            <element>udpv4</element>
                        </transports>
                    </element>
                </value>
            </default_unicast>
            <transport_builtin>
                <mask>UDPv4</mask>
            </transport_builtin>
            <resource_limits>
                <local_writer_allocation>
                    <max_count>1</max_count>
                </local_writer_allocation>
                <local_reader_allocation>
                    <max_count>1</max_count>
                </local_reader_allocation>
                <local_publisher_allocation>
                    <max_count>1</max_count>
                </local_publisher_allocation>
                <local_subscriber_allocation>
                    <max_count>1</max_count>
                </local_subscriber_allocation>
                <local_topic_allocation>
                    <max_count>1</max_count>
                </local_topic_allocation>
                <local_type_allocation>
                    <max_count>1</max_count>
                </local_type_allocation>
                <remote_participant_allocation>
                    <max_count>8</max_count>
                </remote_participant_allocation>
                <remote_writer_allocation>
                    <max_count>8</max_count>
                </remote_writer_allocation>
                <remote_reader_allocation>
                    <max_count>8</max_count>
```

(continues on next page)

(continued from previous page)

```
            </remote_reader_allocation>
            <max_receive_ports>32</max_receive_ports>
            <max_destination_ports>32</max_destination_ports>
        </resource_limits>
    </domain_participant_qos>
    <!-- DataWriter Qos -->
    <datawriter_qos>
        <history>
            <depth>32</depth>
        </history>
        <resource_limits>
            <max_instances>2</max_instances>
            <max_samples>64</max_samples>
            <max_samples_per_instance>32</max_samples_per_instance>
        </resource_limits>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
        <protocol>
            <rtps_reliable_writer>
                <heartbeat_period>
                    <nanosec>250000000</nanosec>
                    <sec>0</sec>
                </heartbeat_period>
            </rtps_reliable_writer>
        </protocol>
        <!-- transports -->
        <unicast>
            <value>
                <element>
                    <transports>
                        <element>udpv4</element>
                    </transports>
                </element>
            </value>
        </unicast>
    </datawriter_qos>
    <!-- DataReader Qos -->
    <datareader_qos>
        <history>
            <depth>32</depth>
        </history>
        <resource_limits>
            <max_instances>2</max_instances>
            <max_samples>64</max_samples>
            <max_samples_per_instance>32</max_samples_per_instance>
        </resource_limits>
        <reliability>
            <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
        <reader_resource_limits>
```

(continues on next page)

(continued from previous page)

```
                <max_remote_writers>10</max_remote_writers>
                <max_remote_writers_per_instance>10</max_remote_writers_per_instance>
            </reader_resource_limits>
            <!-- transports -->
            <unicast>
                <value>
                    <element>
                        <transports>
                            <element>udpv4</element>
                        </transports>
                    </element>
                </value>
            </unicast>
            <multicast>
                <value>
                    <element>
                        <receive_address>127.0.0.1</receive_address>
                        <transports>
                            <element>udpv4</element>
                        </transports>
                    </element>
                </value>
            </multicast>
        </datareader_qos>
    </qos_profile>

    <qos_profile name="DPDEProfile" base_name="DefaultProfile">
        <domain_participant_qos>
            <discovery_config>
                <builtin_discovery_plugins>SDP</builtin_discovery_plugins>
            </discovery_config>
        </domain_participant_qos>
    </qos_profile>

    <qos_profile name="DPSEProfile" base_name="DefaultProfile">
        <domain_participant_qos>
            <discovery_config>
                <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
            </discovery_config>
        </domain_participant_qos>
    </qos_profile>
</qos_library>
</dds>
```

**Note:** *rtiddsmag* only generates code for the QoS policies used by at least one entity, unless the QoS profile has either of the default flags `is_default_participant_factory_profile` or `is_default_qos` set to true.

**Note:** The QoS policies that are generated by *rtiddsmag* are those specified in the XML file.

If a QoS policy is only partially specified, *rtiddsmag* will fill in the missing attributes with the corresponding default values defined in *Connext Micro*

### DomainParticipant Factory QoS

*rtiddsmag* only generates code for the <participant_factory_qos> in the <qos_profile> that has the flag `is_default_participant_factory_profile` set to true. The log verbosity can also be configured by using <verbosity> inside <logging>. For example:

```
<!-- Participant Factory Qos -->
<participant_factory_qos>
    <entity_factory>
        <autoenable_created_entities>false</autoenable_created_entities>
    </entity_factory>
    <resource_limits>
        <max_participants>4</max_participants>
        <max_components>20</max_components>
    </resource_limits>
</participant_factory_qos>
```

*rtiddsmag* generates the following code:

**HelloWorldAppgen.h**

```
#define RTI_APP_GEN___DPF_QOS_QosLibrary_DefaultProfile \
{ \
    {  /* entity_factory */ \
       DDS_BOOLEAN_FALSE /* autoenable_created_entities */ \
    }, \
    {  /* resource_limits */ \
       4L, /* max_participants  */ \
       20L /* max_components */ \
    } \
}
#define RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    ...,
    RTI_APP_GEN___DPF_QOS_QosLibrary_DefaultProfile /* factory_qos */ \
}
```

### DomainParticipant QoS

The example defines a base profile named DefaultProfile, which contains the base QoSs used by each *DomainParticipant*. You can see the content of the *DomainParticipant* QoS below.

```
<domain_participant_qos>
    <discovery>
        <accept_unknown_peers>false</accept_unknown_peers>
```

(continues on next page)

(continued from previous page)

```
        <initial_peers>
            <element>127.0.0.1</element>
            <element>239.255.0.1</element>
        </initial_peers>
        <enabled_transports>
            <element>udpv4</element>
        </enabled_transports>
        <multicast_receive_addresses>
            <element>udpv4://127.0.0.1</element>
            <element>udpv4://239.255.0.1</element>
        </multicast_receive_addresses>
    </discovery>
    <default_unicast>
        <value>
            <element>
                <transports>
                    <element>udpv4</element>
                </transports>
            </element>
        </value>
    </default_unicast>
    <transport_builtin>
        <mask>UDPv4</mask>
    </transport_builtin>
    <resource_limits>
        <local_writer_allocation>
            <max_count>1</max_count>
        </local_writer_allocation>
        <local_reader_allocation>
            <max_count>1</max_count>
        </local_reader_allocation>
        <local_publisher_allocation>
            <max_count>1</max_count>
        </local_publisher_allocation>
        <local_subscriber_allocation>
            <max_count>1</max_count>
        </local_subscriber_allocation>
        <local_topic_allocation>
            <max_count>1</max_count>
        </local_topic_allocation>
        <local_type_allocation>
            <max_count>1</max_count>
        </local_type_allocation>
        <remote_participant_allocation>
            <max_count>8</max_count>
        </remote_participant_allocation>
        <remote_writer_allocation>
            <max_count>8</max_count>
        </remote_writer_allocation>
        <remote_reader_allocation>
            <max_count>8</max_count>
```

(continues on next page)

(continued from previous page)

```
        </remote_reader_allocation>
        <max_receive_ports>32</max_receive_ports>
        <max_destination_ports>32</max_destination_ports>
    </resource_limits>
</domain_participant_qos>
```

This *DomainParticipant* is then inherited by two different profiles, which set up the discovery mechanism:

```
<domain_participant_qos>
    <discovery_config>
        <builtin_discovery_plugins>SDP</builtin_discovery_plugins>
    </discovery_config>
</domain_participant_qos>
<domain_participant_qos>
    <discovery_config>
        <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
    </discovery_config>
</domain_participant_qos>
```

*rtiddsmag* generates the following code for each *DomainParticipant* whose QoS inherits from any of the previous ones, adding those values that are specified in the XML configuration file (which is not the case in our example).

**HelloWorldAppgen.c**

```
const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_initial_peers[2] =
{
    "127.0.0.1",
    "239.255.0.1"
};
const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_discovery_enabled_
↪transports[3] =
{
    "udp1://",
    "udp1://127.0.0.1",
    "udp1://239.255.0.1"
};
const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_transport_enabled_
↪transports[1] =
{
    "udp1"
};
const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_user_traffic_enabled_
↪transports[1] =
{
    "udp1://"
};
```

**HelloWorldAppgen.h**

```
extern const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_initial_peers[2];
extern const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_discovery_enabled_
↪transports[3];
extern const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_transport_enabled_
↪transports[1];
extern const char *const HelloWorldAppLibrary_HelloWorldDPDEPubDP_user_traffic_enabled_
↪transports[1];

#define RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    {   /* entity_factory */ \
        DDS_BOOLEAN_TRUE /* autoenable_created_entities */ \
    }, \
    {   /* discovery */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪initial_peers, 2, 2), /* initial_peers */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪discovery_enabled_transports, 3, 3), /* enabled_transports */ \
        { \
            { { "dpde1" } }, /* RT_ComponentFactoryId_INITIALIZER */ \
                NDDS_Discovery_Property_INITIALIZER \
        }, /* discovery_component */ \
        DDS_BOOLEAN_FALSE /* accept_unknown_peers */ \
    }, \
    {   /* resource_limits  */ \
        1L, /* local_writer_allocation */ \
        1L, /* local_reader_allocation */ \
        1L, /* local_publisher_allocation */ \
        1L, /* local_subscriber_allocation */ \
        1L, /* local_topic_allocation */ \
        1L, /* local_type_allocation */ \
        8L, /* remote_participant_allocation */ \
        8L, /* remote_writer_allocation */ \
        8L, /* remote_reader_allocation */ \
        32L, /* matching_writer_reader_pair_allocation */ \
        32L, /* matching_reader_writer_pair_allocation */ \
        32L, /* max_receive_ports */ \
        32L, /* max_destination_ports */ \
        65536, /* unbound_data_buffer_size */ \
        500UL, /* shmem_ref_transfer_mode_max_segments */ \
        0L, /* participant_user_data_max_length */ \
        DDS_SIZE_AUTO, /* participant_user_data_max_count */ \
        0L, /* topic_data_max_length */ \
        DDS_SIZE_AUTO, /* topic_data_max_count */ \
        0L, /* publisher_group_data_max_length */ \
        DDS_SIZE_AUTO, /* publisher_group_data_max_count */ \
        0L, /* subscriber_group_data_max_length */ \
        DDS_SIZE_AUTO, /* subscriber_group_data_max_count */ \
        0L, /* writer_user_data_max_length */ \
        DDS_SIZE_AUTO, /* writer_user_data_max_count */ \
        0L, /* reader_user_data_max_length */ \
        DDS_SIZE_AUTO, /* reader_user_data_max_count */ \
```

(continues on next page)

(continued from previous page)

```
        64L, /* max_partitions */ \
        256L, /* max_partition_cumulative_characters */ \
        DDS_LENGTH_UNLIMITED, /* max_partition_string_size */ \
        DDS_LENGTH_UNLIMITED /* max_partition_string_allocation */ \
    }, \
    DDS_ENTITY_NAME_QOS_POLICY_DEFAULT, \
    DDS_WIRE_PROTOCOL_QOS_POLICY_DEFAULT, \
    {   /* transports */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪transport_enabled_transports, 1, 1) /* enabled_transports */ \
    }, \
    {   /* user_traffic */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_user_
↪traffic_enabled_transports, 1, 1) /* enabled_transports */ \
    }, \
    DDS_TRUST_QOS_POLICY_DEFAULT, \
    DDS_PROPERTY_QOS_POLICY_DEFAULT, \
    DDS_USER_DATA_QOS_POLICY_DEFAULT \
 }
```

### Publisher QoS

Our example doesn't specify any value for *Publisher* QoS, however *rtiddsmag* would generate code if it was specified.

### DataWriter QoS

The example defines a base profile named DefaultProfile, which contains the base QoSs used by each *DomainParticipant*. You can see the content of the *DataWriter* QoS below.

```
<!-- DataWriter Qos -->
<datawriter_qos>
   <history>
      <depth>32</depth>
   </history>
   <resource_limits>
      <max_instances>2</max_instances>
      <max_samples>64</max_samples>
      <max_samples_per_instance>32</max_samples_per_instance>
    </resource_limits>
    <reliability>
       <kind>RELIABLE_RELIABILITY_QOS</kind>
    </reliability>
    <protocol>
        <rtps_reliable_writer>
           <heartbeat_period>
              <nanosec>250000000</nanosec>
              <sec>0</sec>
```

(continues on next page)

(continued from previous page)

```
            </heartbeat_period>
        </rtps_reliable_writer>
    </protocol>
    <!-- transports -->
    <unicast>
        <value>
            <element>
                <transports>
                    <element>udpv4</element>
                </transports>
            </element>
        </value>
    </unicast>
</datawriter_qos>
```

*rtiddsmag* generates the following code:

### HelloWorldAppgen.c

```
const char *const
HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_HelloWorldDPDEDW_transport_
↪enabled_transports[1] =
{
   "udp1://"
};
```

### HelloWorldAppgen.h

```
extern const char *const
HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_HelloWorldDPDEDW_transport_
↪enabled_transports[1];

#define RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_
↪HelloWorldDPDEDW \
{ \
    DDS_DEADLINE_QOS_POLICY_DEFAULT, \
    DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
    {   /* history */ \
        DDS_KEEP_LAST_HISTORY_QOS, /* kind */ \
        32L /* depth */ \
    }, \
    {   /* resource_limits */ \
        64L, /* max_samples */ \
        2L, /* max_instances */ \
        32L /* max_samples_per_instance */ \
    }, \
    DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
    DDS_OWNERSHIP_STRENGTH_QOS_POLICY_DEFAULT, \
    DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
    {   /* reliability */ \
        DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
        {   /* max_blocking_time */ \
```

(continues on next page)

(continued from previous page)

```
            0L, /* sec */ \
            100000000L /* nanosec */ \
        } \
    }, \
    DDS_DURABILITY_QOS_POLICY_DEFAULT, \
    DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
    DDS_TRANSPORT_ENCAPSULATION_QOS_POLICY_DEFAULT, \
    DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT, \
    {   /* protocol */ \
        DDS_RTPS_AUTO_ID, /* rtps_object_id */ \
        { /* rtps_reliable_writer */ \
        {   /* heartbeat_period */ \
            0L, /* sec */ \
            250000000L /* nanosec */ \
        },  \
        1L, /* heartbeats_per_max_samples */ \
        DDS_LENGTH_UNLIMITED, /* max_send_window */ \
        DDS_LENGTH_UNLIMITED, /* max_heartbeat_retries */ \
        {   /* first_write_sequence_number */ \
            0, /* high */ \
            1  /* low */ \
        } \
    }, \
    DDS_BOOLEAN_TRUE /* serialize_on_write */ \
    }, \
    DDS_TYPESUPPORT_QOS_POLICY_DEFAULT, \
    {   /* transports */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪HelloWorldDPDEPub_HelloWorldDPDEDW_transport_enabled_transports, 1, 1) /* enabled_
↪transports */ \
    }, \
    RTI_MANAGEMENT_QOS_POLICY_DEFAULT, \
    DDS_DATAWRITERRESOURCE_LIMITS_QOS_POLICY_DEFAULT, \
    DDS_PUBLISH_MODE_QOS_POLICY_DEFAULT, \
    DDS_USER_DATA_QOS_POLICY_DEFAULT, \
    DDS_DATAWRITERQOS_APPGEN_INITIALIZER \
    NULL, \
    DDS_DataWriterTransferModeQosPolicy_INITIALIZER \
}
```

### Subscriber QoS

Our example doesn't specify any value for Subscriber QoS, however *rtiddsmag* would generate code if it was specified.

### DataReader QoS

The example defines a base profile named DefaultProfile, which contains the base QoSs used by each *DomainParticipant*. You can see the content of the *DataReader* QoS below.

```
<!-- DataReader QoS -->
<datareader_qos>
    <history>
        <depth>32</depth>
    </history>
    <resource_limits>
        <max_instances>2</max_instances>
        <max_samples>64</max_samples>
        <max_samples_per_instance>32</max_samples_per_instance>
    </resource_limits>
    <reliability>
        <kind>RELIABLE_RELIABILITY_QOS</kind>
    </reliability>
    <reader_resource_limits>
        <max_remote_writers>10</max_remote_writers>
        <max_remote_writers_per_instance>10</max_remote_writers_per_instance>
    </reader_resource_limits>
    <!-- transports -->
    <unicast>
        <value>
            <element>
                <transports>
                    <element>udpv4</element>
                </transports>
            </element>
        </value>
    </unicast>
    <multicast>
        <value>
            <element>
                <receive_address>127.0.0.1</receive_address>
                <transports>
                    <element>udpv4</element>
                </transports>
            </element>
        </value>
    </multicast>
</datareader_qos>
```

*rtiddsmag* generates the following code:

**HelloWorldAppgen.c**

```
const char *const
HelloWorldAppLibrary_HelloWorldDPDESubDP_HelloWorldDPDESub_HelloWorldDPDEDR_transport_
↪enabled_transports[2] =
{
   "udp1://",
   "udp1://127.0.0.1"
};
```

### HelloWorldAppgen.h

```
extern const char *const
HelloWorldAppLibrary_HelloWorldDPDESubDP_HelloWorldDPDESub_HelloWorldDPDEDR_transport_
↪enabled_transports[2];
#define RTI_APP_GEN___DR_QOS_HelloWorldAppLibrary_HelloWorldDPDESubDP_HelloWorldDPDESub_
↪HelloWorldDPDEDR \
{ \
    DDS_DEADLINE_QOS_POLICY_DEFAULT, \
    DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
    {   /* history */ \
        DDS_KEEP_LAST_HISTORY_QOS, /* kind */ \
        32L /* depth */ \
    }, \
    {   /* resource_limits */ \
        64L, /* max_samples */ \
        2L, /* max_instances */ \
        32L /* max_samples_per_instance */ \
    }, \
    DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
    DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
    {   /* reliability */ \
        DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
        {   /* max_blocking_time */ \
            0L, /* sec */ \
            0L /* nanosec */ \
        } \
    }, \
    DDS_DURABILITY_QOS_POLICY_DEFAULT, \
    DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
    DDS_TRANSPORT_ENCAPSULATION_QOS_POLICY_DEFAULT, \
    DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT, \
    DDS_TYPESUPPORT_QOS_POLICY_DEFAULT, \
    DDS_DATA_READER_PROTOCOL_QOS_POLICY_DEFAULT, \
    {   /* transports */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDESubDP_
↪HelloWorldDPDESub_HelloWorldDPDEDR_transport_enabled_transports, 2, 2) /* enabled_
↪transports */ \
    }, \
    {   /* reader_resource_limits */ \
        10L, /* max_remote_writers */ \
        10L, /* max_remote_writers_per_instance */ \
        1L, /* max_samples_per_remote_writer */ \
        1L, /* max_outstanding_reads */ \
```

(continues on next page)

(continued from previous page)

```
        DDS_NO_INSTANCE_REPLACEMENT_QOS, /* instance_replacement */ \
        4L, /* max_routes_per_writer */ \
        DDS_MAX_AUTO, /* max_fragmented_samples */ \
        DDS_MAX_AUTO, /* max_fragmented_samples_per_remote_writer */ \
        DDS_SIZE_AUTO /* shmem_ref_transfer_mode_attached_segment_allocation */ \
    }, \
    RTI_MANAGEMENT_QOS_POLICY_DEFAULT,  \
    DDS_USER_DATA_QOS_POLICY_DEFAULT, \
    DDS_DATAREADERQOS_APPGEN_INITIALIZER \
    NULL \
}
```

### Topic QoS

Our example doesn't specify any value for Topic QoS; however, *rtiddsmag* would generate code if it were specified.

### Transport and Discovery Configuration

*rtiddsmag* creates the code necessary to configure each one of the available transports used by *Connext Micro* (UDP, SHMEM, and Zero Copy v2) and the discovery mechanism (Dynamic and Static discovery). It also generates the name automatically for each component regardless of if it is a transport or discovery; for this *rtiddsmag* will add a *DomainParticipant* number at the end of its name, only if that configuration is not used by any other *DomainParticipant* with the exception of the UDP transport, which will always have the same name regardless of the *DomainParticipant* number:

- UDP Transport: `udpv4`.
- SHMEM Transport: `shmem` + participant_number.
- Zero Copy v2 Transport: `zcopy` + participant_number.
- DPDE: `dpde` + participant_number.
- DPSE: `dpse` + participant_number.

These names can be changed by using the `...Name` options described in *MAG Command-Line Options*.

---

**Note:**

- *rtiddsmag* will only create the transport configuration based on the strongly typed XML elements in the schema. *rtiddsmag* **will not** use the values in the property tag to configure the transport.
- If the length of one of these names exceeds the maximum length, *rtiddsmag* will throw an error.

- `-incrementUdpName` can be used to add the `participant_number` suffix to the UDP transport name if needed.

---

The following configuration specifies dynamic discovery:

```
<domain_participant_qos>
    <discovery_config>
        <builtin_discovery_plugins>SDP</builtin_discovery_plugins>
    </discovery_config>
</domain_participant_qos>
```

**HelloWorldAppgen.h**

```
#define RTI_APP_GEN___dpde__HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde1 \
{ \
    RT_ComponentFactoryProperty_INITIALIZER, /* _parent */ \
    {   /*participant_liveliness_assert_period */ \
        30L, /* sec */ \
        0L /* nanosec */ \
    }, \
    {   /*participant_liveliness_lease_duration */ \
        100L, /* sec */ \
        0L /* nanosec */ \
    }, \
    5, /* initial_participant_announcements */ \
    {   /*initial_participant_announcement_period */ \
        1L, /* sec */ \
        0L /* nanosec */ \
    }, \
    DDS_BOOLEAN_FALSE, /* cache_serialized_samples */ \
    DDS_LENGTH_AUTO, /* max_participant_locators */ \
    4, /* max_locators_per_discovered_participant */ \
    8, /* max_samples_per_builtin_endpoint_reader */ \
    DDS_LENGTH_UNLIMITED, /* builtin_writer_max_heartbeat_retries */ \
    {   /*builtin_writer_heartbeat_period */ \
        0L, /* sec */ \
        100000000L /* nanosec */ \
    }, \
    1L /* builtin_writer_heartbeats_per_max_samples */ \
    DDS_PARTICIPANT_MESSAGE_READER_RELIABILITY_KIND_INITIALIZER \
}

#define RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    ...
    {   /* discovery */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪initial_peers, 2, 2), /* initial_peers */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪discovery_enabled_transports, 3, 3), /* enabled_transports */ \
        { \
            { { "dpde1" } }, /* RT_ComponentFactoryId_INITIALIZER */ \
```

(continues on next page)

(continued from previous page)

```
                NDDS_Discovery_Property_INITIALIZER \
        }, /* discovery_component */ \
        DDS_BOOLEAN_FALSE /* accept_unknown_peers */ \
    }, \
    ...
}
```

**Note:**

- *rtiddsmag* will generate an error if the list of available transports for the *DomainParticipant*, *DataWriter*, and *DataReader* contains a transport alias that is not part of the `transport_builtin` mask.
- *rtiddsmag* will not generate code for the SHMEM or UDPv4 transport if it is not specified in the `transport_builtin` mask.
- UDP transformation is not supported in XML.

When using the transport alias to specify the `enabled_transports` for the discovery *DomainParticipant*, *DataWriter* or *DataReader*, you could use the transport names for the built-in transport plugins: `shmem`, `udpv4`, and `zcopy`. *rtiddsmag* will automatically modify this alias to match the new one with the *DomainParticipant* number at the end of the name.

The Zero Copy v2 transport is configured differently than the other transports. It cannot be configured through the `transport_builtin` element in XML, and it cannot be enabled with the `transport_builtin` mask. Instead, it can be configured through properties in the XML file. The following properties are required by *rtiddsmag* to configure the Zero Copy v2 transport:

- dds.transport.micro.zero_copy.max_samples_per_notif
- dds.transport.micro.zero_copy.user_intf[1]
- dds.transport.micro.zero_copy.user_property[2]

The following additional properties are only required if you are using the default implementation of the notification mechanism for Zero Copy v2; see *Register the Zero Copy v2 transport* for more information. When configuring the `user_intf` in Zero Copy v2, you must define all or none of the values. If only some of them are defined, MAG will report an error and the generated code will not work with *Connext Micro*.

- dds.transport.micro.zero_copy.user_property.intf_addr
- dds.transport.micro.zero_copy.user_property.thread_prop
  - dds.transport.micro.zero_copy.user_property.thread_prop.stack_size
  - dds.transport.micro.zero_copy.user_property.thread_prop.priority
  - dds.transport.micro.zero_copy.user_property.thread_prop.options

[1] This property is only required if you choose to implement your own notification mechanism and not use the default implementation provided by RTI.

[2] Resolves to ZCOPY_NotifMechanismProperty when using the default notification mechanism.

- dds.transport.micro.zero_copy.user_property.max_receive_ports
- dds.transport.micro.zero_copy.user_property.max_routes

The following additional properties are only required if you are using your own notification mechanism for Zero Copy v2, NOT the default implementation.

- dds.transport.micro.zero_copy.user_intf.create_instance
- dds.transport.micro.zero_copy.user_intf.delete_instance
- dds.transport.micro.zero_copy.user_intf.get_route_table
- dds.transport.micro.zero_copy.user_intf.reserve_address
- dds.transport.micro.zero_copy.user_intf.release_address
- dds.transport.micro.zero_copy.user_intf.resolve_address
- dds.transport.micro.zero_copy.user_intf.add_route
- dds.transport.micro.zero_copy.user_intf.delete_route
- dds.transport.micro.zero_copy.user_intf.bind
- dds.transport.micro.zero_copy.user_intf.unbind
- dds.transport.micro.zero_copy.user_intf.send
- dds.transport.micro.zero_copy.user_intf.notify_recv_port
- dds.transport.micro.zero_copy.user_intf.create_instance

The following code is an example of how to configure the Zero Copy v2 transport in XML and the resulting code that *rtiddsmag* generates:

```
<property>
   <value>
       <element>
           <name>dds.micro.zero_copy.enable</name>
           <value>true</value>
       </element>
       <element>
           <name>dds.micro.zero_copy.max_samples_per_notif</name>
           <value>256</value>
       </element>
       <element>
           <name>dds.micro.zero_copy.user_property.intf_addr</name>
           <value>125</value>
       </element>
   </value>
</property>
```

**HelloWorldAppgen.c**

```
const struct ZCOPY_NotifMechanismProperty zcopy1_user_property = RTI_APP_GEN___zcopy__
↪HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1_NOTIF_USER_PROPERTY;
const struct ZCOPY_NotifLoaderFactoryProperty HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪zcopy[1] =
{
    RTI_APP_GEN___zcopy__HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1
};
const struct ComponentFactoryRegisterModel HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪register_components[3] =
{
    /* … */
    {
        "zcopy1_", /* register_name */
        ZCOPY_Loader_get_interface, /* register_intf */
        &HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy[0]._parent._parent._parent, /*␣
↪register_property */
        NULL /* register_listener */
    }
};
```

### HelloWorldAppgen.h

```
#define RTI_APP_GEN___zcopy__HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1_NOTIF_USER_
↪PROPERTY \
{ \
    125U, /* intf_addr */ \
    OSAPI_ThreadProperty_INITIALIZER, \
    2U, /* max_receive_ports */ \
    32U /* max_routes */ \
}


extern const struct ZCOPY_NotifMechanismProperty zcopy1_user_property;


#define RTI_APP_GEN___zcopy__HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1_PROPERTY \
{ \
    NETIO_InterfaceFactoryProperty_INITIALIZER, \
    256L, /* max_samples_per_notif */ \
    NULL, /* user_intf */ \
    &zcopy1_user_property /* user_property */ \
}


#define RTI_APP_GEN___zcopy__HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1 \
{ \
    RTI_APP_GEN___zcopy__HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1_PROPERTY, /* _
↪parent */ \
    "zcopy1" /* notif_transport_name */ \
}
```

### UDP Transport Configuration

The following UDP transport properties must be configured via the PROPERTY QoS policy for the *DomainParticipant*:

- disable_multicast_bind
- disable_multicast_interface_select
- transport_priority_mapping_low
- transport_priority_mapping_high
- transport_priority_mask
- max_unicast_send_socket

The names of these properties should use the prefix `dds.transport.UDPv4.builtin.`:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>dds.transport.UDPv4.builtin.disable_multicast_bind</name>
                <value>TRUE</value>
            </element>
            <element>
                <name>dds.transport.UDPv4.builtin.disable_multicast_interface_select</
↪name>
                <value>FALSE</value>
            </element>
            <element>
                <name>dds.transport.UDPv4.builtin.transport_priority_mapping_low</name>
                <value>0</value>
            </element>
        </value>
    </property>
</participant_qos>
```

Most other UDP transport properties can be configured directly as XML tags under `<transport_builtin><udpv4>`:

```
<transport_builtin>
    <mask>UDPv4</mask>
    <udpv4>
        <ignore_loopback_interface>0</ignore_loopback_interface>
        <allow_interfaces_list>
            <element>eth0</element>
            <element>eth1</element>
        </allow_interfaces_list>
        <recv_socket_buffer_size>65535</recv_socket_buffer_size>
        <send_socket_buffer_size>65535</send_socket_buffer_size>
        <multicast_ttl>100</multicast_ttl>
    </udpv4>
</transport_builtin>
```

The following table provides a comprehensive reference of all UDP transport configuration properties:

Table 3.13: UDP Transport Configuration Properties

| Property Name | Configuration Method | Description | Default Value | MAG Support |
|---|---|---|---|---|
| allow_interface | XML Tag via allow_interfaces_list | String sequence of network interface names (e.g., "en0", "eth1") that are allowed for communication. Empty sequence allows all interfaces. | Empty (all allowed) | Yes |
| deny_interface | XML Tag via deny_interfaces_list | String sequence of network interface names that are NOT allowed for communication. Checked after allow_interface list. Empty sequence denies no interfaces. | Empty (none denied) | Yes |
| max_send_buffer_size | XML Tag via send_socket_buffer_size | Maximum size of the send socket buffer. Must be at least as large as the largest sample. Typically a multiple of maximum concurrent samples. | 256 KB | Yes |
| max_receive_buffer_size | XML Tag via recv_socket_buffer_size | Maximum size of the receive socket buffer. Must be at least as large as the largest sample. Typically a multiple of maximum concurrent samples. | 256 KB | Yes |
| max_message_size | XML Tag via message_size_max | Maximum size of the message which can be received, including packet overhead. | 8192 bytes | Yes |
| multicast_ttl | XML Tag | Multicast Time-To-Live value. Limits the number of hops a packet can pass through before being dropped. Used only for multicast. | 1 | Yes |
| nat | Cannot be configured through MAG | Network Address Translation table for NAT traversal. Requires programmatic configuration via `UDP_NatEntrySeq_*` functions. Limited use; only supports static private-to-public address mapping without hole punching. | Empty | No |
| if_table | XML Tag via interface_table | Manual configuration of available network interfaces. Used when IP stack does not support reading interface lists. Requires programmatic configuration via `UDP_InterfaceTable_add_entry`. | {(0,0,0)} | Yes |
| multicast_interface | XML Tag | Specific network interface to use for sending multicast packets. Leave unset for OS to select. | Any (OS selects) | Yes |
| is_default_interface | Constant and cannot be configured | Flag indicating whether this UDP transport is used if no other transport is found. | TRUE | Yes (hard-coded TRUE) |
| dis-able_auto_interface_config | XML Tag | Prevents UDP transport from reading the interface list on platforms that support it. Set to TRUE to disable auto-detection. | FALSE | Yes |

The following UDP transport properties are defined by *Connext Micro* but are not currently configurable by MAG:

| Property Name | Why MAG Does Not Consume It | Notes |
|---|---|---|
| nat | Requires explicit programmatic NAT entry construction via C API; MAG does not model these entries. | Micro supports it, MAG does not expose it. |
| is_default_interface | The template emits this as `RTI_TRUE` and does not read a model value for it. | Present in generated C, not user-configurable through MAG. |
| enable_interface_bind | The template emits this as `RTI_FALSE` and does not read a model value for it. | Present in generated C, not user-configurable through MAG. |
| source_rules | MAG does not model UDP transformation rules. | Transformation support not implemented. |
| destination_rules | MAG does not model UDP transformation rules. | Transformation support not implemented. |
| transform_udp_mode | Depends on transformation support that MAG does not implement. | Transformation support not implemented. |
| transform_locator_kind | Depends on transformation support that MAG does not implement. | Transformation support not implemented. |

### Shared Memory Transport (SHMEM) Configuration

*rtiddsmag* supports configuring the `dds.transport.minimum_compatibility_version` property, which you can set via the PROPERTY QoS policy for the *DomainParticipant*. Refer to *SHMEM Configuration* for more information on `dds.transport.minimum_compatibility_version`.

### Discovery Configuration

*rtiddsmag* supports configuring the following DISCOVERY QoS policy fields:

- `dds.micro.discovery.enable_participant_discovery_by_name`
- `dds.micro.discovery.enable_endpoint_discovery_queue`

However, you must configure these fields via the PROPERTY QoS policy instead of DISCOVERY, as shown in the example code below:

```
<property>
   <value>
       <element>
           <name>dds.micro.discovery.enable_participant_discovery_by_name</name>
           <value>true</value>
       </element>
       <element>
           <name>dds.micro.discovery.enable_endpoint_discovery_queue</name>
```

(continues on next page)

(continued from previous page)

```
            <value>true</value>
        </element>
    </value>
</property>
```

For more information on these fields, refer to *DomainParticipant Discovery by Name* and *[Major] Possible mismatched endpoint messages during discovery*.

### Flow Controllers

*rtiddsmag* creates code which will be used by *Connext Micro* to create a custom flow controller. The custom flow controller is configured through properties in the XML file. Let's see an example of how to configure a custom flow controller named `custom_flowcontroller` and the code that *rtiddsmag* generates:

```
<domain_participant_qos>
    ...
    <property>
        <value>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.max_tokens
                </name>
                <value>2</value>
            </element>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.tokens_added_per_
↪period
                </name>
                <value>2</value>
            </element>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.tokens_leaked_per_
↪period
                </name>
                <!-- The value -1 means LENGTH_UNLIMITED -->
                <value>-1</value>
            </element>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.period.sec
                </name>
                <value>0</value>
            </element>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.period.nanosec
                </name>
```

(continues on next page)

(continued from previous page)

```
                <value>100000000</value>
            </element>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.bytes_per_token
                </name>
                <value>1024</value>
            </element>
        </value>
    </property>
</domain_participant_qos>

<datawriter_qos>
    <publish_mode>
        <flow_controller_name>
            dds.flow_controller.token_bucket.custom_flowcontroller
        </flow_controller_name>
        <kind>ASYNCHRONOUS_PUBLISH_MODE_QOS</kind>
        <priority>12</priority>
    </publish_mode>
</datawriter_qos>
```

### HelloWorldAppgen.c

```
const struct APPGEN_CustomFlowControllerModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_flow_controllers[1] =
{
    {
        "custom_flowcontroller", /* name */
        RTI_APP_GEN___FC_P_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_custom_
↪flowcontroller /* flow_controller_property */
    }
};
```

### HelloWorldAppgen.h

```
#define
RTI_APP_GEN___FC_P_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_custom_flowcontroller \
{ \
    NETIO_FlowControllerProperty_INITIALIZER, \
    DDS_EDF_FLOW_CONTROLLER_SCHED_POLICY, /* scheduling_policy */ \
    { /* token_bucket */ \
        2L, /* max_tokens */ \
        2L, /* tokens_added_per_period */ \
        -1L, /* tokens_leaked_per_period */ \
        { /* period */ \
            0L, /* sec */ \
            100000000L /* nanosec */ \
        }, \
        1024L /* bytes_per_token */ \
    }, \
```

(continues on next page)

(continued from previous page)

```
    DDS_BOOLEAN_FALSE /* is_vendor_specific */ \
}

#define
RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_
↪HelloWorldDPDEDW \
{ \
    ...
    {   /* publish_mode */ \
        DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS, /* max_remote_readers */ \
        "custom_flowcontroller", /* flow_controller_name */ \
        12L /* priority */ \
    }, \
    ...
}

extern const struct APPGEN_CustomFlowControllerModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_flow_controllers[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    ...
    1UL, /* custom_flow_controller_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_flow_controllers /* custom_flow_controllers␣
↪*/ \
}
```

The three built-in Flow Controllers are also supported by *rtiddsmag*:

- DEFAULT_FLOW_CONTROLLER_NAME
- FIXED_RATE_FLOW_CONTROLLER_NAME
- ON_DEMAND_FLOW_CONTROLLER_NAME

The generated code is slightly different when any of these three built-in Flow Controllers are configured, as there is no need to generate code to register the Flow Controller.

```
<datawriter_qos>
    <publish_mode>
        <flow_controller_name>DEFAULT_FLOW_CONTROLLER_NAME</flow_controller_name>
        <kind>ASYNCHRONOUS_PUBLISH_MODE_QOS</kind>
        <priority>12</priority>
    </publish_mode>
</datawriter_qos>
```

**HelloWorldAppgen.h**

```
#define
RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_
↪HelloWorldDPDEDW \
{ \
```

(continues on next page)

(continued from previous page)

```
    ...
    {   /* publish_mode */ \
        DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS, /* max_remote_readers */ \
        "DDS_DEFAULT_FLOW_CONTROLLER_NAME", /* flow_controller_name */ \
        12L /* priority */ \
    }, \
    ...
}

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    ...
    0UL, /* custom_flow_controller_count */ \
    NULL /* custom_flow_controllers */ \
}
```

**Note:** A flow controller is only used by Micro when the publish_mode kind is set to either *ASYNCHRONOUS_PUBLISH_MODE_QOS* or *AUTOMATIC_PUBLISH_MODE_QOS*.

### Static Discovery

*rtiddsmag* iterates through each *DomainParticipant* definition in the XML configuration file, creating the remote entities that are needed to communicate with applications that use static discovery, and updating the `object_id` of each *DataWriter* or *DataReader* involved if they don't have a valid value or they are using the default value.

Let's see an example of two applications that use static discovery and how *rtiddsmag* generates the necessary code that will be asserted by *Connext Micro* to communicate with both applications:

```
<domain_participant name="HelloWorldDPSEPubDP"
    domain_ref="HelloWorldLibrary::HelloWorldDomain">
    <publisher name="HelloWorldDPSEPub">
        <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPSEDW">
            <datawriter_qos base_name="QosLibrary::DPSEProfile"/>
        </data_writer>
    </publisher>
    <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
</domain_participant>

<domain_participant name="HelloWorldDPSESubDP"
    domain_ref="HelloWorldLibrary::HelloWorldDomain">
    <subscriber name="HelloWorldDPSESub">
        <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPSEDR">
          <datareader_qos base_name="QosLibrary::DPSEProfile"/>
        </data_reader>
    </subscriber>
    <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
</domain_participant>
```

For these two *DomainParticipants*, *rtiddsmag* will update the `rtps_object_id` for the *DataWriter* and *DataReader*, since they didn't have any values set in the XML file. You can see this in the following snippet from **HelloWorldAppgen.h**:

```
#define
RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldDPSEPub_
↪HelloWorldDPSEDW \
{ \
    ... 
    {   /* protocol */ \
        1UL, /* rtps_object_id */ \
        { /* rtps_reliable_writer */ \
            {   /* heartbeat_period */ \
                0L, /* sec */ \
                250000000UL /* nanosec */ \
            },  \
            1L, /* heartbeats_per_max_samples */ \
            DDS_LENGTH_UNLIMITED, /* max_send_window */ \
            DDS_LENGTH_UNLIMITED, /* max_heartbeat_retries */ \
            {   /* first_write_sequence_number */ \
                0, /* high */ \
                1  /* low */ \
            } \
        }, \
        DDS_BOOLEAN_TRUE /* serialize_on_write */ \
    }, \
    ... 
}

#define
RTI_APP_GEN___DR_QOS_HelloWorldAppLibrary_HelloWorldDPSESubDP_HelloWorldDPSESub_
↪HelloWorldDPSEDR \
{ \
    ... 
    {   /* protocol */ \
        2UL /* rtps_object_id */ \
    }, \
    ... 
}
```

*rtiddsmag* will also generate the remote *DomainParticipants*, *DataWriters*, and *DataReaders* that need to be asserted in order for endpoints to match:

**HelloWorldAppgen.c**

```
const struct APPGEN_RemoteSubscriptionModel
HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_subscribers[1] =
{
    RTI_APP_GEN__RSD_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_HelloWorldDPSESub_HelloWorldDPSEDR
};

const struct APPGEN_RemoteParticipantModel
```

(continues on next page)

(continued from previous page)

```
HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_participants[1] =
{
    {
        "HelloWorldDPSESubDP", /* name */
        0UL,  /* remote_publisher_count */
        NULL, /* remote_publishers */
        1UL,  /* remote_subscriber_count */
        HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_subscribers /* remote_
↪subscribers */
    }
};

const struct APPGEN_RemotePublicationModel
HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_publishers[1] =
{
    RTI_APP_GEN__RPD_HelloWorldAppLibrary_HelloWorldDPSESubDP_HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_HelloWorldDPSEPub_HelloWorldDPSEDW
};

const struct APPGEN_RemoteParticipantModel
HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_participants[1] =
{
    {
        "HelloWorldDPSEPubDP", /* name */
        1UL, /* remote_publisher_count */
        HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_publishers, /* remote_publishers␣
↪*/
        0UL, /* remote_subscriber_count */
        NULL /* remote_subscribers */
    }
};
```

### HelloWorldAppgen.h

```
#define RTI_APP_GEN__RSD_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_HelloWorldDPSESub_HelloWorldDPSEDR \
{ \
    { /* subscription_data */ \
        { \
            { 0, 0, 0, 2 } /* key */ \
        }, \
        { \
            { 0, 0, 0, 0 } /* participant_key */ \
        }, \
        "HelloWorldTopic", /* topic_name */ \
        "HelloWorldType",  /* type_name */ \
        DDS_DEADLINE_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
        DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
        {   /* reliability */ \
            DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
```

(continues on next page)

(continued from previous page)

```
            {   /* max_blocking_time */ \
                0L, /* sec */ \
                0L  /* nanosec */ \
            } \
        }, \
        DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
        DDS_DURABILITY_QOS_POLICY_DEFAULT, \
        DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT \
    }, \
    HelloWorldTypePlugin_get /* get_type_plugin */ \
}
extern const struct APPGEN_RemoteSubscriptionModel HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_remote_subscribers[1];
extern const struct APPGEN_RemoteParticipantModel HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_remote_participants[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPSEPubDP \
{ \
    "HelloWorldDPSEPubDP", /* name */ \
    RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPSEPubDP, /* domain_participant_
↪factory */ \
    RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPSEPubDP, /* domain_participant_
↪qos */ \
    0L,  /* domain_id */ \
    1UL, /* type_registration_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_type_registrations, /* type_registrations */
↪ \
    1UL, /* topic_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_topics, /* topics */ \
    1UL, /* publisher_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_publishers, /* publishers */ \
    0UL,  /* subscriber_count */ \
    NULL, /* subscribers */ \
    1UL,  /* remote_participant_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_participants /* remote_participants␣
↪*/ \
    0UL,  /* custom_flow_controller_count */ \
    NULL, /* custom_flow_controllers */ \
}

#define RTI_APP_GEN__RPD_HelloWorldAppLibrary_HelloWorldDPSESubDP_HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_HelloWorldDPSEPub_HelloWorldDPSEDW \
{ \
    { /* publication_data */ \
        { \
            { 0, 0, 0, 1 } /* key */ \
        }, \
        { \
```

(continues on next page)

(continued from previous page)

```
            { 0, 0, 0, 0 } /* participant_key */ \
        }, \
        "HelloWorldTopic", /* topic_name */ \
        "HelloWorldType",  /* type_name */ \
        DDS_DEADLINE_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_STRENGTH_QOS_POLICY_DEFAULT, \
        DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
        {   /* reliability */ \
            DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
            {   /* max_blocking_time */ \
                0L, /* sec */ \
                100000000L /* nanosec */ \
            } \
        }, \
        DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
        DDS_DURABILITY_QOS_POLICY_DEFAULT, \
        DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT \
    }, \
    HelloWorldTypePlugin_get /* get_type_plugin */ \
}

extern const struct APPGEN_RemotePublicationModel HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_remote_publishers[1];
extern const struct APPGEN_RemoteParticipantModel HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_remote_participants[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPSESubDP \
{ \
    "HelloWorldDPSESubDP", /* name */ \
    RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPSESubDP, /* domain_participant_
↪factory */ \
    RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPSESubDP, /* domain_participant_
↪qos */ \
    0L,  /* domain_id */ \
    1UL, /* type_registration_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_type_registrations, /* type_registrations */
↪ \
    1UL, /* topic_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_topics, /* topics */ \
    0UL,  /* publisher_count */ \
    NULL, /* publishers */ \
    1UL,  /* subscriber_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_subscribers, /* subscribers */ \
    1UL, /* remote_participant_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_participants /* remote_participants␣
↪*/ \
    0UL, /* custom_flow_controller_count */ \
    NULL /* custom_flow_controllers */ \
```

(continues on next page)

(continued from previous page)

```
#define RTI_APP_GEN__RSD_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_HelloWorldDPSESub_HelloWorldDPSEDR \
{ \
    { /* subscription_data */ \
        { \
            { 0, 0, 0, 2 } /* key */ \
        }, \
        { \
            { 0, 0, 0, 0 } /* participant_key */ \
        }, \
        "HelloWorldTopic", /* topic_name */ \
        "HelloWorldType",  /* type_name */ \
        DDS_DEADLINE_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
        DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
        {   /* reliability */ \
            DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
            {   /* max_blocking_time */ \
                0L, /* sec */ \
                0L  /* nanosec */ \
            } \
        }, \
        DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
        DDS_DURABILITY_QOS_POLICY_DEFAULT, \
        DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT \
    }, \
    HelloWorldTypePlugin_get /* get_type_plugin */ \
}
extern const struct APPGEN_RemoteSubscriptionModel HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_remote_subscribers[1];
extern const struct APPGEN_RemoteParticipantModel HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_remote_participants[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPSEPubDP \
{ \
    "HelloWorldDPSEPubDP", /* name */ \
    RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPSEPubDP, /* domain_participant_
↪factory */ \
    RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPSEPubDP, /* domain_participant_
↪qos */ \
    0L,  /* domain_id */ \
    1UL, /* type_registration_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_type_registrations, /* type_registrations */
↪ \
    1UL, /* topic_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_topics, /* topics */ \
    1UL, /* publisher_count */ \
```

(continues on next page)

(continued from previous page)

```
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_publishers, /* publishers */ \
    0UL,  /* subscriber_count */ \
    NULL, /* subscribers */ \
    1UL,  /* remote_participant_count */ \
    HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_participants /* remote_participants
↪*/ \
    0UL,  /* custom_flow_controller_count */ \
    NULL, /* custom_flow_controllers */ \
}

#define RTI_APP_GEN__RPD_HelloWorldAppLibrary_HelloWorldDPSESubDP_HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_HelloWorldDPSEPub_HelloWorldDPSEDW \
{ \
    { /* publication_data */ \
        { \
            { 0, 0, 0, 1 } /* key */ \
        }, \
        { \
            { 0, 0, 0, 0 } /* participant_key */ \
        }, \
        "HelloWorldTopic", /* topic_name */ \
        "HelloWorldType",  /* type_name */ \
        DDS_DEADLINE_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_STRENGTH_QOS_POLICY_DEFAULT, \
        DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
        {   /* reliability */ \
            DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
            {   /* max_blocking_time */ \
                0L, /* sec */ \
                100000000L /* nanosec */ \
            } \
        }, \
        DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
        DDS_DURABILITY_QOS_POLICY_DEFAULT, \
        DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT \
    }, \
    HelloWorldTypePlugin_get /* get_type_plugin */ \
}

extern const struct APPGEN_RemotePublicationModel HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_remote_publishers[1];
extern const struct APPGEN_RemoteParticipantModel HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_remote_participants[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPSESubDP \
{ \
    "HelloWorldDPSESubDP", /* name */ \
    RTI_APP_GEN__DPF_HelloWorldAppLibrary_HelloWorldDPSESubDP, /* domain_participant_
↪factory */ \
```

(continues on next page)

(continued from previous page)

```
    RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPSESubDP, /* domain_participant_
↪qos */ \
    0L,  /* domain_id */ \
    1UL, /* type_registration_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_type_registrations, /* type_registrations */
↪ \
    1UL, /* topic_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_topics, /* topics */ \
    0UL,  /* publisher_count */ \
    NULL, /* publishers */ \
    1UL,  /* subscriber_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_subscribers, /* subscribers */ \
    1UL, /* remote_participant_count */ \
    HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_participants /* remote_participants␣
↪*/ \
    0UL, /* custom_flow_controller_count */ \
    NULL /* custom_flow_controllers */ \
}
```

### Lightweight Security Plugin

*rtiddsmag* can generate code to configure the *Lightweight Security Plugin* for *Connext Micro*. You can configure the plugin through properties in the XML file.

*rtiddsmag* requires the following properties to configure the plugin:

- `dds.sec.crypto.enable`
- `dds.sec.crypto.rtps_psk_secret_passphrase`
- `dds.sec.crypto.rtps_psk_symmetric_cipher_algorithm`

The following properties are optional:

- `dds.micro.transform.get_interface_func`[3]
- `dds.micro.transform.name`
- `dds.micro.transform.err_listener.callback`[4]
- `dds.micro.transform.err_listener.user_ptr`[4]
- `dds.micro.pass_tracker.get_interface_func`[5]
- `dds.micro.pass_tracker.polling_interval_ms`[6]
- `dds.micro.pass_tracker.enable_init_read`[6]

[3] If `dds.micro.transform.get_interface_func` is not set, the default `PSK_OSSL_get_interface` is used.

[4] This property is only supported when `dds.micro.transform.get_interface_func` is not set, or is set to `PSK_OSSL_get_interface`.

[5] If `dds.micro.pass_tracker.get_interface_func` is not set, the default `PSK_FilePassTracker_get_interface` is used.

[6] This property is only supported when `dds.micro.pass_tracker.get_interface_func` is not set, or is set to `PSK_FilePassTracker_get_interface`:

- `dds.micro.pass_tracker.err_listener.callback`[6]
- `dds.micro.pass_tracker.err_listener.user_ptr`[6]
- `dds.sec.access.rtps_psk_protection_kind`
- `com.rti.serv.secure.cryptography.max_blocks_per_session`

*rtiddsmag* automatically generates the name of the pre-shared key (PSK) component. *rtiddsmag* also appends a *DomainParticipant* number to the name, as long as the particular configuration is not being used by any other *DomainParticipant*:

```
psk + participant_number
```

For example, the first unique pre-shared key configuration would be named `psk1`; the second, `psk2`; and so on.

This name can be changed with the `-pskName` option described in *MAG Command-Line Options*.

**Note:** If the length of the name exceeds the maximum length, *rtiddsmag* will throw an error.

Refer to the following code snippet as an example of how to configure the *Lightweight Security Plugin* in XML and the resulting code that *rtiddsmag* generates:

```
<property>
  <value>
      <element>
          <name>dds.micro.crypto.enable</name>
          <value>true</value>
      </element>
      <element>
          <name>dds.sec.crypto.rtps_psk_secret_passphrase</name>
          <value>data:,404166165:castle super radar denial swing lunar kind swarm wet␣
↪toilet output harbor basic begin margin huge year visit</value>
      </element>
      <element>
          <name>dds.sec.crypto.rtps_psk_symmetric_cipher_algorithm</name>
          <value>AUTO</value>
      </element>
      <element>
          <name>dds.sec.access.rtps_psk_protection_kind</name>
          <value>ENCRYPT</value>
      </element>
      <element>
          <name>com.rti.serv.secure.cryptography.max_blocks_per_session</name>
          <value>12</value>
      </element>
      <element>
          <name>dds.micro.transform.get_interface_func</name>
          <value>PSK_OSSL_get_interface</value>
      </element>
      <element>
```

(continues on next page)

(continued from previous page)

```
            <name>dds.micro.transform.name</name>
            <value>myTransf</value>
        </element>
        <element>
            <name>dds.micro.transform.err_listener.callback</name>
            <value>myCallback</value>
        </element>
        <element>
            <name>dds.micro.transform.err_listener.user_ptr</name>
            <value>myUserPtr</value>
        </element>
        <element>
            <name>dds.micro.pass_tracker.get_interface_func</name>
            <value>PSK_FilePassTracker_get_interface</value>
        </element>
        <element>
            <name>dds.micro.pass_tracker.polling_interval_ms</name>
            <value>250</value>
        </element>
        <element>
            <name>dds.micro.pass_tracker.enable_init_read</name>
            <value>false</value>
        </element>
        <element>
            <name>dds.micro.pass_tracker.err_listener.callback</name>
            <value>myPassTrackerCallback</value>
        </element>
        <element>
            <name>dds.micro.pass_tracker.err_listener.user_ptr</name>
            <value>myPassTrackerUserPtr</value>
        </element>
    </value>
</property>
```

### HelloWorldAppgen.c

```
const struct DDS_PskServiceFactoryProperty HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪psk[1] =
{
    RTI_APP_GEN___psk__HelloWorldAppLibrary_HelloWorldDPDEPubDP_psk1
};
const struct ComponentFactoryRegisterModel HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪register_components[2] =
{
    /* … */
    {
        "psk1", /* register_name */
        DDS_PskServiceFactory_get_interface, /* register_intf */
        &HelloWorldAppLibrary_HelloWorldDPDEPubDP_psk[0]._parent, /* register_property */
        NULL /* register_listener */
    }
};
```

**HelloWorldAppgen.h**

```
extern const struct PSK_ErrListener psk1_configuration_err_listener;
extern const struct DDS_SecTransform_Configuration psk1_configuration;

extern const struct PSK_FilePassTrackerErrListener psk1_pass_tracker_configuration_err_
↪listener;
extern const struct PSK_PassTrackerConfiguration psk1_pass_tracker_configuration;

#define RTI_APP_GEN___psk__HelloWorldAppLibrary_HelloWorldDPDEPubDP_psk1 \
{ \
    RT_ComponentFactoryProperty_INITIALIZER, \
    "psk1", /* service_name */ \
    &PSK_OSSL_get_interface, /* psl_get_interface_func */ \
    &psk1_configuration, /* psl_config */ \
    &PSK_FilePassTracker_get_interface, /* pass_tracker_get_interface_func */ \
    &psk1_pass_tracker_configuration /* pass_tracker_config */ \
}

#define RTI_APP_GEN___psk__HelloWorldAppLibrary_HelloWorldDPDEPubDP_psk1_PASS_TRACKER_
↪CONFIG \
{ \
    250, /* polling_interval_ms */ \
    RTI_FALSE, /* enable_init_read */ \
    RTI_APP_GEN___psk__HelloWorldAppLibrary_HelloWorldDPDEPubDP_psk1_PASS_TRACKER_CONFIG_
↪ERROR_LISTENER /* err_listener */ \
}

#define RTI_APP_GEN___psk__HelloWorldAppLibrary_HelloWorldDPDEPubDP_psk1_PASS_TRACKER_
↪CONFIG_ERROR_LISTENER \
{ \
    myPassTrackerCallback, /* callback */ \
    &myPassTrackerUserPtr /* user_ptr */ \
}

extern const struct DDS_PskServiceFactoryProperty HelloWorldAppLibrary_
↪HelloWorldDPDEPubDP_psk[1];
```

### Content filtering

*rtiddsmag* can configure *Content Filtering* with syntax that is compatible with *Connext Professional* application configuration.

### Configuring DataReaders

Unlike in manual configuration, we don't configure content filters in the *DataReader*'s QoS. Instead, we configure it on the *DataReader* itself. Refer to the following code snippet for an example *DataReader* configuration and the code that *rtiddsmag* generates:

```
<subscriber name="HelloWorldDPDESub">
    <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPDEDR">
        <datareader_qos base_name="QosLibrary::DPDEProfile"/>
        <content_filter name="myFilter1" kind="builtin.sql">
            <expression>id == 1</expression>
        </content_filter>
    </data_reader>
</subscriber>
```

**Note:** When using *rtiddsmag*, the only supported filter kind is `builtin.sql`. The syntax is the same as when configuring a `DDS_ContentFilterQosPolicy`.

### HelloWorldAppgen.h

```
#define RTI_APP_GEN___DR_QOS_HelloWorldAppLibrary_HelloWorldDPDESubDP_HelloWorldDPDESub_
↪HelloWorldDPDEDR \
{ \
    DDS_DEADLINE_QOS_POLICY_DEFAULT, \
    DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
    {   /* history */ \
        DDS_KEEP_LAST_HISTORY_QOS, /* kind */ \
        32L /* depth */ \
    }, \
    {   /* resource_limits */ \
        64L, /* max_samples */ \
        2L, /* max_instances */ \
        32L /* max_samples_per_instance */ \
    }, \
    DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
    DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
    {   /* reliability */ \
        DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
        {   /* max_blocking_time */ \
            0L, /* sec */ \
            0L /* nanosec */ \
        } \
    }, \
    DDS_DURABILITY_QOS_POLICY_DEFAULT, \
    DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
    DDS_TRANSPORT_ENCAPSULATION_QOS_POLICY_DEFAULT, \
    DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT, \
    DDS_TYPESUPPORT_QOS_POLICY_DEFAULT, \
    DDS_DATA_READER_PROTOCOL_QOS_POLICY_DEFAULT, \
    {   /* transports */ \
```

(continues on next page)

(continued from previous page)

```
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDESubDP_
↪HelloWorldDPDESub_HelloWorldDPDEDR_transport_enabled_transports, 2, 2) /* enabled_
↪transports */ \
    }, \
    {   /* reader_resource_limits */ \
        10L, /* max_remote_writers */ \
        10L, /* max_remote_writers_per_instance */ \
        1L, /* max_samples_per_remote_writer */ \
        1L, /* max_outstanding_reads */ \
        DDS_NO_INSTANCE_REPLACEMENT_QOS, /* instance_replacement */ \
        4L, /* max_routes_per_writer */ \
        DDS_MAX_AUTO, /* max_fragmented_samples */ \
        DDS_MAX_AUTO, /* max_fragmented_samples_per_remote_writer */ \
        DDS_SIZE_AUTO /* shmem_ref_transfer_mode_attached_segment_allocation */ \
    }, \
    RTI_MANAGEMENT_QOS_POLICY_DEFAULT,  \
    DDS_USER_DATA_QOS_POLICY_DEFAULT, \
    DDS_PROPERTY_QOS_POLICY_DEFAULT, \
    { /* content_filter */ \
        "myFilter1", /* filter_name */ \
        "DDSSQL", /* filter_class_name */ \
        "id == 1", /* filter_expression */ \
        DDS_SEQUENCE_INITIALIZER /* expression_parameters */ \
    }, \
    DDS_DATAREADERQOS_APPGEN_INITIALIZER \
    NULL \
}
```

**HelloWorldAppgen.c**

```
const struct ComponentFactoryRegisterModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_register_components[3] =
{
    {
        "dpde1", /* register_name */
        DPDE_DiscoveryFactory_get_interface, /* register_intf */
        &HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde[0]._parent, /* register_property␣
↪*/
        NULL /* register_listener */
    },
    {
        "udp1", /* register_name */
        UDP_InterfaceFactory_get_interface, /* register_intf */
        &HelloWorldAppLibrary_HelloWorldDPDEPubDP_udpv4[0]._parent._parent, /* register_
↪property */
        NULL /* register_listener */
    },
    {
        DDS_FILTER_PLUGIN_FACTORY_DEFAULT_NAME, /* register_name */
        DDS_FilterPluginFactory_get_interface, /* register_intf */
        NULL, /* register_property */
```

(continues on next page)

(continued from previous page)

```
        NULL /* register_listener */
    },
};

/* Companion variable: only emitted when the QoS XML specifies
 * dds.filter.micro.sql.predicate_max_count_per_expression and does
 * not explicitly set dds.filter.micro.disable_builtin_sql_filter=true.
 */
#if DDS_ENABLE_SQL_FILTER
static const struct DDS_SqlFilterProperty
    RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_sql_filter_prop =
{
    8L
};
#endif /* DDS_ENABLE_SQL_FILTER */
```

### Configuring DomainParticipants

To enable filtering support on a specific *DomainParticipant*, set the following property in the *DomainParticipant*'s QoS:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>dds.filter.micro.enable</name>
                <value>true</value>
            </element>
        </value>
    </property>
</participant_qos>
```

When filtering support is enabled on any *DomainParticipant*, the filter plugin factory will be registered automatically. Unlike manual configuration, enabling filter support for a single *DomainParticipant* does not automatically enable it for all *DomainParticipants*; the property must be set to `true` for each *DomainParticipant* that wishes to use the feature.

The following optional properties are supported on the *DomainParticipant*. These options are all equivalent to the fields of the `DDS_FilterQosPolicy`.

| Property | Type |
|---|---|
| dds.filter.micro.enable[7] | Boolean |
| dds.filter.micro.disable_writer_filtering | Boolean |
| dds.filter.micro.disable_builtin_sql_filter | Boolean |
| dds.filter.micro.resource_limits.<br>filter_class_max_length | DDS_Long [1, 15] |
| dds.filter.micro.resource_limits.<br>filter_class_max_count | DDS_Long [1, 2147483647] |
| dds.filter.micro.resource_limits.<br>filter_expression_max_length | DDS_Long [1, 255] |
| dds.filter.micro.resource_limits.<br>filter_expression_max_count | DDS_Long [1, 2147483646] or DDS_LENGTH_AUTO |
| dds.filter.micro.resource_limits.<br>filter_parameter_max_count_per_expression | DDS_Long [0, 16] |
| dds.filter.micro.resource_limits.<br>filter_parameter_max_length | DDS_Long [0, 2147483646] |
| dds.filter.micro.sql.predicate_max_count_per_expression | DDS_Long [1, 65535], default: 4 |

**Note:** When dds.filter.micro.sql.predicate_max_count_per_expression is set in the QoS XML (at any value) and dds.filter.micro.disable_builtin_sql_filter is not explicitly set to true, *rtiddsmag* automatically sets disable_builtin_sql_filter to true in the generated DDS_FilterQosPolicy and then registers the builtin SQL filter manually with the configured DDS_SqlFilterProperty. If dds.filter.micro.disable_builtin_sql_filter is explicitly true, the filter is not registered and the predicate count has no effect.

Refer to the following code snippet for an example *DomainParticipant* configuration and the code that *rtiddsmag* generates:

```
<!-- Content filtering properties -->
    <element>
        <name>dds.filter.micro.disable_writer_filtering</name>
        <value>true</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.disable_builtin_sql_filter</name>
        <value>true</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.sql.predicate_max_count_per_expression</name>
        <value>8</value>
        <propagate>false</propagate>
    </element>
```

(continues on next page)

[7] This is an optional property, because by default *rtiddsmag* will automatically generate code to register the component in the *DomainParticipantFactory* if there is at least one content filter used by one *DataReader*.

(continued from previous page)

```
    <element>
        <name>dds.filter.micro.resource_limits.filter_class_max_length</name>
        <value>7</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.resource_limits.filter_class_max_count</name>
        <value>1</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.resource_limits.filter_expression_max_length</name>
        <value>63</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.resource_limits.filter_expression_max_count</name>
        <value>LENGTH_AUTO</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.resource_limits.filter_parameter_max_count_per_expression
↪</name>
        <value>4</value>
        <propagate>false</propagate>
    </element>
    <element>
        <name>dds.filter.micro.resource_limits.filter_parameter_max_length</name>
        <value>15</value>
        <propagate>false</propagate>
    </element>
```

**HelloWorldAppgen.c**

```
#define RTI_APP_GEN___DP_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    {   /* entity_factory */ \
        DDS_BOOLEAN_TRUE /* autoenable_created_entities */ \
    }, \
    {   /* discovery */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪initial_peers, 2, 2), /* initial_peers */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪discovery_enabled_transports, 3, 3), /* enabled_transports */ \
        { \
            { { "dpde1" } }, /* RT_ComponentFactoryId_INITIALIZER */ \
                NDDS_Discovery_Property_INITIALIZER \
        }, /* discovery_component */ \
        DDS_BOOLEAN_FALSE /* accept_unknown_peers */ \
    }, \
    {   /* resource_limits  */ \
```

(continues on next page)

(continued from previous page)

```
        1L, /* local_writer_allocation */ \
        1L, /* local_reader_allocation */ \
        1L, /* local_publisher_allocation */ \
        1L, /* local_subscriber_allocation */ \
        1L, /* local_topic_allocation */ \
        1L, /* local_type_allocation */ \
        8L, /* remote_participant_allocation */ \
        8L, /* remote_writer_allocation */ \
        8L, /* remote_reader_allocation */ \
        32L, /* matching_writer_reader_pair_allocation */ \
        32L, /* matching_reader_writer_pair_allocation */ \
        32L, /* max_receive_ports */ \
        32L, /* max_destination_ports */ \
        65536, /* unbound_data_buffer_size */ \
        500UL, /* shmem_ref_transfer_mode_max_segments */ \
        0L, /* participant_user_data_max_length */ \
        DDS_SIZE_AUTO, /* participant_user_data_max_count */ \
        0L, /* topic_data_max_length */ \
        DDS_SIZE_AUTO, /* topic_data_max_count */ \
        0L, /* publisher_group_data_max_length */ \
        DDS_SIZE_AUTO, /* publisher_group_data_max_count */ \
        0L, /* subscriber_group_data_max_length */ \
        DDS_SIZE_AUTO, /* subscriber_group_data_max_count */ \
        0L, /* writer_user_data_max_length */ \
        DDS_SIZE_AUTO, /* writer_user_data_max_count */ \
        0L, /* reader_user_data_max_length */ \
        DDS_SIZE_AUTO, /* reader_user_data_max_count */ \
        64L, /* max_partitions */ \
        256L, /* max_partition_cumulative_characters */ \
        DDS_LENGTH_UNLIMITED, /* max_partition_string_size */ \
        DDS_LENGTH_UNLIMITED /* max_partition_string_allocation */ \
    }, \
    DDS_ENTITY_NAME_QOS_POLICY_DEFAULT, \
    DDS_WIRE_PROTOCOL_QOS_POLICY_DEFAULT, \
    {   /* transports */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪transport_enabled_transports, 1, 1) /* enabled_transports */ \
    }, \
    {   /* user_traffic */ \
        REDA_StringSeq_INITIALIZER_W_LOAN(HelloWorldAppLibrary_HelloWorldDPDEPubDP_user_
↪traffic_enabled_transports, 1, 1) /* enabled_transports */ \
    }, \
    DDS_TRUST_QOS_POLICY_DEFAULT, \
    DDS_PROPERTY_QOS_POLICY_DEFAULT, \
    DDS_USER_DATA_QOS_POLICY_DEFAULT \
    ,{ /* filter */ \
        DDS_FILTER_PLUGIN_FACTORY_DEFAULT_ID, /* name */ \
        { /* resource_limits */ \
            7L, /* filter_class_max_length */ \
            2L, /* filter_class_max_count */ \
            63L, /* filter_expression_max_length */ \
```

(continues on next page)

(continued from previous page)

```
            DDS_LENGTH_AUTO, /* filter_expression_max_count */ \
            4L, /* filter_parameter_max_count_per_expression */ \
            15L /* filter_parameter_max_length */ \
        }, \
        DDS_BOOLEAN_TRUE, /* disable_writer_filtering */ \
        DDS_BOOLEAN_TRUE /* disable_builtin_sql_filter */ \
    } \
}
```

### 3.17.5 Generating AUTOSAR Applications

This section demonstrates how to use deployment-aware code generation features to create AUTOSAR-compatible DDS applications. The process involves:

1. Defining a complete system in DDS system XML format.
2. Running `rtiddsmag` with `-deployment` and `-applicationType` selectors.

   > **Attention:** To generate AUTOSAR code, `-deployment` and `-applicationType` *must be specified together*. If `-deployment` is not specified, the output will only include `Appgen.c` and `Appgen.h` files. If `-applicationType` is missing, the output will default to `Micro4`, which is not yet fully supported.

3. Running `rtiarcgen` to generate AUTOSAR type ARXML from DDS XML types. `rtiarcgen` will also internally call `rtiddsgen` in the same execution to generate the DDS Type support files.

   > **Attention:** We highly recommend letting `rtiarcgen` call `rtiddsgen` by passing the `-ddsTypeSupport` option to `rtiarcgen` instead of calling `rtiddsgen` separately. Calling `rtiddsgen` separately from `rtiarcgen` can cause AUTOSAR types and DDS types to have conflicting names.

4. Generating AUTOSAR code from an AUTOSAR toolchain (we use Vector DaVinci in examples) to a specific location to utilize already-implemented runnables.

* `rtiarcgen` can be configured to call `rtiddsgen` by adding `-ddsTypeSupport` to the command. We recommend using this option to generate DDS type support files for AUTOSAR workflows.

** MAG creates and populates some runnables with basic functionalities. Note that DaVinci toolkits may add new, empty runnables or rename existing ones.

*** For more details about the interaction between DaVinci toolkits and the generated runnables, refer to *AUTOSAR runnables*.

We'll go through these steps in more detail in the following subsections.

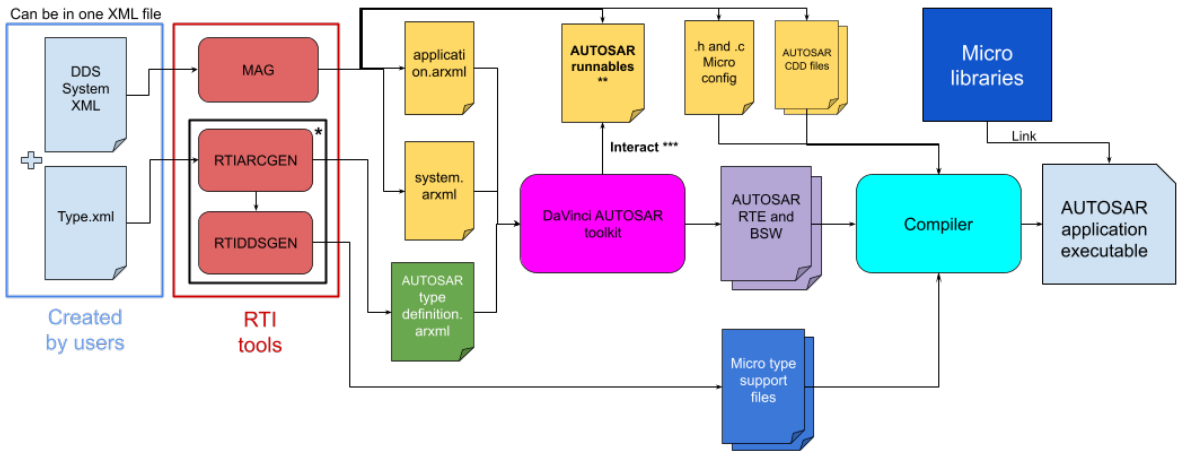



Figure 3.9: Workflow for converting XML files into AUTOSAR application code

### Define your system in DDS system XML

The DDS System XML format allows you to define the complete topology of your distributed system, including topics, participants, nodes, and deployment configurations. The following example demonstrates a one-node deployment of the `hello_world` application in an AUTOSAR environment. The application is organized as follows:

```
Deployment Library
   Deployment Scenario
       Deployment
           Applications
               Application
                   Domain Participant
                       Publisher
                          DataWriter HelloWorldTopic
                       Subscriber
                          DataReader HelloWorldTopic
```

This XML structure organizes the system into five key libraries:

- **Domain Library**: Defines DDS topics and associated data types used for communication.
- **Participant Library**: Groups domain participants (publishers and subscribers) with their QoS policy configuration.
- **Application Library**: Wraps *DomainParticipants* into logical deployable applications.
- **Node Library**: Represents target deployment nodes (e.g., ECUs, processors, and devices).
- **Deployment Library**: Maps applications to nodes and specifies deployment-specific configuration such as memory heaps and AUTOSAR resource ranges.

The XML schema is located at `resource/schema/dds-xml_system_definitions.xsd`. Use this in your XML files with:

```
xsi:noNamespaceSchemaLocation="resource/schema/dds-xml_system_definitions.xsd"
```

The following example code demonstrates a complete DDS System XML configuration with deployment sections, based on the DDS System use case:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="resource/schema/dds-xml_system_definitions.xsd">
    <types>
        <struct name= "HelloWorld">
            <member name="initialFirstName" type="char8"/>
            <member name="initialLastName" type="char8"/>
        </struct>
    </types>

    <qos_library name="QosLibrary">
        <qos_profile name="DefaultProfile" is_default_participant_factory_profile="true">
            <!-- Participant Factory Qos -->
            <participant_factory_qos>
                <entity_factory>
                    <autoenable_created_entities>false</autoenable_created_entities>
                </entity_factory>
            </participant_factory_qos>
            <!-- Participant Qos -->
            <participant_qos>
                <discovery>
                    <accept_unknown_peers>false</accept_unknown_peers>
                    <initial_peers>
                        <element>127.0.0.1</element>
                    </initial_peers>
                    <enabled_transports>
                        <element>udpv4</element>
                    </enabled_transports>
                </discovery>
                <default_unicast>
                    <value>
                        <element>
                            <transports>
                                <element>udpv4</element>
                            </transports>
                        </element>
                    </value>
                </default_unicast>
                <resource_limits>
                    <local_writer_allocation>
                        <max_count>1</max_count>
                    </local_writer_allocation>
                    <local_reader_allocation>
                        <max_count>1</max_count>
                    </local_reader_allocation>
                    <local_publisher_allocation>
                        <max_count>1</max_count>
                    </local_publisher_allocation>
```

(continues on next page)

(continued from previous page)

```
                    <local_subscriber_allocation>
                        <max_count>1</max_count>
                    </local_subscriber_allocation>
                    <local_topic_allocation>
                        <max_count>1</max_count>
                    </local_topic_allocation>
                    <local_type_allocation>
                        <max_count>1</max_count>
                    </local_type_allocation>
                    <remote_participant_allocation>
                        <max_count>8</max_count>
                    </remote_participant_allocation>
                    <remote_writer_allocation>
                        <max_count>8</max_count>
                    </remote_writer_allocation>
                    <remote_reader_allocation>
                        <max_count>8</max_count>
                    </remote_reader_allocation>
                    <max_receive_ports>32</max_receive_ports>
                    <max_destination_ports>32</max_destination_ports>
                </resource_limits>
                <transport_builtin>
                    <mask>UDPv4</mask>
                    <udpv4>
                        <disable_auto_interface_config>true</disable_auto_interface_
↪config>
                    </udpv4>
                </transport_builtin>
                <discovery_config>
                    <builtin_discovery_plugins>SDP</builtin_discovery_plugins>
                    <initial_participant_announcements>5</initial_participant_
↪announcements>
                    <max_initial_participant_announcement_period>
                        <sec>1</sec>
                        <nanosec>0</nanosec>
                    </max_initial_participant_announcement_period>
                    <participant_liveliness_assert_period>
                        <sec>10</sec>
                        <nanosec>0</nanosec>
                    </participant_liveliness_assert_period>
                    <participant_liveliness_lease_duration>
                        <sec>30</sec>
                        <nanosec>0</nanosec>
                    </participant_liveliness_lease_duration>
                </discovery_config>

            </participant_qos>
            <!-- DataWriter Qos -->
            <datawriter_qos>
                <history>
                    <depth>32</depth>
```

(continues on next page)

(continued from previous page)

```
            </history>
            <resource_limits>
                <max_instances>2</max_instances>
                <max_samples>64</max_samples>
                <max_samples_per_instance>32</max_samples_per_instance>
            </resource_limits>
            <reliability>
                <kind>RELIABLE_RELIABILITY_QOS</kind>
            </reliability>
            <protocol>
                <rtps_reliable_writer>
                    <heartbeat_period>
                        <nanosec>250000000</nanosec>
                        <sec>0</sec>
                    </heartbeat_period>
                </rtps_reliable_writer>
            </protocol>
        </datawriter_qos>
        <!-- DataReader Qos -->
        <datareader_qos>
            <history>
                <depth>32</depth>
            </history>
            <resource_limits>
                <max_instances>2</max_instances>
                <max_samples>64</max_samples>
                <max_samples_per_instance>32</max_samples_per_instance>
            </resource_limits>
            <reliability>
                <kind>RELIABLE_RELIABILITY_QOS</kind>
            </reliability>
            <reader_resource_limits>
                <max_remote_writers>10</max_remote_writers>
                <max_remote_writers_per_instance>10</max_remote_writers_per_instance>
            </reader_resource_limits>
        </datareader_qos>
    </qos_profile>
    <qos_profile name="CddProfile" base_name="DefaultProfile">
        <participant_qos>
            <transport_builtin>
                <!-- Set correct interface name(s)-->
                <udpv4>
                    <ignore_loopback_interface>1</ignore_loopback_interface>
                    <interface_table>
                        <element>
                            <interface_name>ethernet</interface_name>
                            <address>10.10.65.152</address>
                            <net_mask>255.255.0.0</net_mask>
                            <flags>
                                UDP_INTERFACE_INTERFACE_UP_FLAG</flags>
                        </element>
```

(continues on next page)

(continued from previous page)

```
                    </interface_table>
                    <allow_interfaces_list>
                        <element>ethernet</element>
                    </allow_interfaces_list>
                </udpv4>
            </transport_builtin>
        </participant_qos>
    </qos_profile>
</qos_library>

<!-- Domain Library -->
<domain_library name="DomainLibrary">
    <domain name="myDomain" domain_id="0">
        <register_type name="HelloWorld" type_ref="HelloWorld" />

        <topic name="HelloWorldTopic" register_type_ref="HelloWorld" />
    </domain>
</domain_library>

<!-- Participant Library -->
<domain_participant_library name="AppLibrary">
    <domain_participant name="MCU_DDS_CDD"
        domain_ref="DomainLibrary::myDomain">
        <publisher name="MCU_DDS_CDD_Pub">
            <data_writer topic_ref="HelloWorldTopic" name="MCU_DDS_CDD_HelloWorld_DW
↪">
                <datawriter_qos base_name="QosLibrary::CddProfile" />
            </data_writer>
        </publisher>
        <subscriber name="MCU_DDS_CDD_Sub">
            <data_reader topic_ref="HelloWorldTopic" name="MCU_DDS_CDD_HelloWorld_DR
↪">
                <datareader_qos base_name="QosLibrary::CddProfile" />
            </data_reader>
        </subscriber>
        <participant_qos base_name="QosLibrary::CddProfile" />
    </domain_participant>
</domain_participant_library>

<!-- Added manually to utilize deployment features -->
<application_library name="MyApplicationLib">
    <application name="MyApplication" isCert="false" >
        <domain_participant name="MCU_DDS_CDD" base_name="AppLibrary::MCU_DDS_CDD" />
    </application>
</application_library>

<node_library name="MyNodeLib">
    <node name="MyNode1"/>
    <node name="MyNode2"/>
</node_library>
```

(continues on next page)

(continued from previous page)

```
    <deployment_library name="MyDeploymentLib">
        <deployment_scenario name="MyDeploymentScenario">
            <deployment name="MyDeployment1">
                <node node_ref="MyNodeLib::MyNode1" />
                <applications>
                    <application name="AutosarApp" application_ref=
↪"MyApplicationLib::MyApplication" />
                </applications>
                <configuration>
                    <autosar>
                        <heap>
                            <element>
                                <size>240</size>
                                <start_address>0x60000100</start_address>
                                <name>Heap_area_1</name>
                            </element>
                            <element>
                                <size>96</size>
                                <start_address>0x50000100</start_address>
                                <name>Heap_area_2</name>
                            </element>
                            <element>
                                <size>96</size>
                                <start_address>0x40000100</start_address>
                                <name>Heap_area_3</name>
                            </element>
                            <element>
                                <size>96</size>
                                <start_address>0x30000100</start_address>
                                <name>Heap_area_4</name>
                            </element>
                            <element>
                                <size>96</size>
                                <start_address>0x10000100</start_address>
                                <name>Heap_area_5</name>
                            </element>
                        </heap>
                        <mutex_resource_id>MutexResource1</mutex_resource_id>
                        <max_local_addr_id>2</max_local_addr_id>
                        <send_local_addr_id>2</send_local_addr_id>
                    </autosar>
                </configuration>
            </deployment>
        </deployment_scenario>
    </deployment_library>
</dds>
```

### Run MAG with deployment and application type selectors

After creating your DDS System XML file, use `rtiddsmag` to generate deployment-aware code for a specific deployment target. The `-deployment` option selects the deployment from your XML, and the `-applicationType AUTOSAR_CDD` option generates AUTOSAR CDD configuration and adapter artifacts.

Both options are required when using the deployment-aware workflow. `rtiddsmag` will generate the following:

- Layer-specific implementations (DDS layer, adapter layer, AUTOSAR configuration).
- Deployment-specific code organization matching the selected deployment.
- Multi-deployment artifacts if you run `rtiddsmag` multiple times with different `-deployment` values.

Run the following command with your XML and deployment name:

Linux

```
$RTIMEHOME/bin/rtiddsmag -inputXml hello_world.xml -deployment MyDeployment1 -
↪applicationType AUTOSAR_CDD -language C -replace
```

macOS

```
$RTIMEHOME/bin/rtiddsmag -inputXml hello_world.xml -deployment MyDeployment1 -
↪applicationType AUTOSAR_CDD -language C -replace
```

Windows

```
%RTIMEHOME%\bin\rtiddsmag.bat -inputXml hello_world.xml -deployment MyDeployment1 -
↪applicationType AUTOSAR_CDD -language C -replace
```

When you run this command, `rtiddsmag` generates code with the following directory structure:

```
MyDeploymentLib/
  MyDeploymentScenario/
      MyDeployment1/
          AutosarApp/
              MCU_DDS_CDD/
                  adaptation /
                      dds_cdd_adapter.h
                      dds_cdd_adapter.c
                  autosar_gen /
                      Autosar runnables DdsCdd.c
                  autosar_model /
                      DdsCddType.arxml
                  dds_impl /
                      dds_impl.h
                      dds_impl.c
                      *Appgen.h
                      *Appgen.c
```

(continues on next page)

(continued from previous page)

```
                MCU_DDS_CDD_EXAMPLE_ASWC /
                    autosar_model /
                        application.arxml
```

The generated code is organized hierarchically by deployment scenario and deployment name. This structure enables maintaining separate code artifacts for different ECUs without file name conflicts. The generated files will be located in `<deployment_library_name>/<scenario_name>/<deployment_name>/<application_name>`. We'll refer to this directory as `<MAG_output_folder>` in the documentation going forward.

---

**Note:** Refer to *DDS CDD configuration* for instructions on how to configure the DDS CDD for use on MicroSAR platforms.

---

### Generate AUTOSAR type ARXML with rtiarcgen

After running `rtiddsmag` for AUTOSAR integration, you must generate AUTOSAR type definitions from your DDS XML types using `rtiarcgen`. More information on `rtiarcgen` can be found in the AUTOSAR Runtime CDD Code Generator User's Manual. This section will only cover the commands and configuration necessary for this workflow.

First, we recommend running the following commands as a pre-check:

```
rtiarcgen -help
rtiarcgen -selfTest
```

The `-selfTest` command validates template access and checks optional integration with `rtiddsgen`. Resolve any template path issues before proceeding.

Create the `dds_gen` directory at <MAG_output_folder>/<Domain_Participant_name>/

Next, use the following commands to generate AUTOSAR type ARXML from your DDS XML:

Linux

```
cd <MAG_output_folder>/<Domain_Participant_name>/dds_gen
export RTIDDSGEN_PATH=<Micro Installation Path>/rtiddsgen/scripts/rtiddsgen
export RTIDDSGEN_PARAMS="-create typefiles -micro -language C"

rtiarcgen -arxmlTypes "$SYSTEM_XML" -arxmlPath . -singleFile -target "$TARGET" -
↪ddsTypeSupport -conversions
```

macOS

```
cd <MAG_output_folder>/<Domain_Participant_name>/dds_gen
export RTIDDSGEN_PATH=<Micro Installation Path>/rtiddsgen/scripts/rtiddsgen
export RTIDDSGEN_PARAMS="-create typefiles -micro -language C"

rtiarcgen -arxmlTypes "$SYSTEM_XML" -arxmlPath . -singleFile -target "$TARGET" -
↪ddsTypeSupport -conversions
```

(continues on next page)

(continued from previous page)

Windows

```
cd <MAG_output_folder>/<Domain_Participant_name>/dds_gen
set RTIDDSGEN_PATH=<Micro Installation Path>\rtiddsgen\scripts\rtiddsgen.bat
set RTIDDSGEN_PARAMS="-create typefiles -micro -language C"

rtiarcgen.bat -arxmlTypes "%SYSTEM_XML%" -arxmlPath . -singleFile -target "%TARGET%" -
↪ddsTypeSupport -conversions
```

If needed, you can convert the type IDL to XML with the following command:

Linux

```
rtiddsgen -convertToXml ../$TYPE_IDL
```

macOS

```
rtiddsgen -convertToXml ../$TYPE_IDL
```

Windows

```
rtiddsgen.bat -convertToXml ../$TYPE_IDL
```

Expected output:

- One AUTOSAR type file (typically `<input_basename>_types.arxml`) in `<MAG_output_folder>/<Domain_Participant_name>/dds_gen`
- DDS type support files:
  - `<input_basename>_types.h` and `<input_basename>_types.c`
  - `<input_basename>_typesPlugin.h` and `<input_basename>_typesPlugin.c`
  - `<input_basename>_typesSupport.h` and `<input_basename>_typesSupport.c`
- DDS AUTOSAR type conversion files (`<input_basename>_conversions.h` and `<input_basename>_conversions.c`)

**Note:** When using `rtiarcgen`, input mode is inferred from the first file extension. For this step, provide DDS XML (`.xml`) input files only, not mixed ARXML+XML inputs.

#### Import both ARXML outputs into AUTOSAR toolchain

After running `rtiarcgen` and `rtiddsmag`, import both ARXML outputs into your AUTOSAR authoring environment. For Vector DaVinci, use this order:

1. **Import type definitions first**: import the ARXML generated by `rtiarcgen` (for example, `dds_gen/<input_basename>_types.arxml`).
2. **Import deployment/configuration second**: import MAG ARXML output from `<MAG_output_folder>/<participant>/autosar_model/DdsCddType.arxml`.
3. **Import application configuration**: import MAG second ARXML output from `<MAG_output_folder>/<participant>__EXAMPLE_ASWC/autosar_model/application.arxml`.

Importing in this order ensures that type references used by the MAG-generated AUTOSAR configuration are already present in the DaVinci project.

We recommend the following Vector DaVinci workflow:

1. Open your target ECU DaVinci project (DaVinci Developer/Configurator).
2. Import the `rtiarcgen` type ARXML into the project data type package.
3. Run model validation and resolve any namespace/package merge prompts.
4. Import the MAG-generated AUTOSAR configuration ARXML.
5. Re-run validation and confirm that all type references resolve.
6. Regenerate RTE/BSW artifacts according to your ECU build flow.

If your system contains multiple deployments, repeat `rtiddsmag` generation per deployment and import each deployment-specific `rtiddsmag` ARXML after the shared type ARXML has already been imported.

The generated code is organized in deployment-specific directories that isolate each deployment's artifacts. This structure supports multi-deployment scenarios where a single system XML may generate code for multiple target deployments (e.g., separate code for publisher and subscriber ECUs).

For each deployment, `rtiddsmag` generates three layers:

1. **DDS Layer**
   - `dds_impl/`: Contains the MAG-generated C code for DDS configuration (*DomainParticipant* initialization, QoS settings, heaps, memory management.
   - `dds_gen/`: Core DDS type support code; serialization and deserialization plugins, type definitions, transport support, and type conversion between DDS and AUTOSAR. This folder also contains the AUTOSAR type definition ARXML file. These files are portable across AUTOSAR environments and implement the DDS wire protocol.
2. **Adapter Layer**

- `adaptation/`: Bridges the DDS layer and AUTOSAR application code. Adapters provide AUTOSAR-specific interfaces to DDS participants, publishers, and subscribers. This layer isolates your application from DDS implementation details.

3. **AUTOSAR Configuration and Runnables**

   - `autosar_model/`: Contains AUTOSAR System Description ARXML. This file is intended for an AUTOSAR toolchain integration. It defines the AUTOSAR application structure, including components, ports, and interfaces, based on your DDS System XML configuration.

- `autosar_gen/`: Generated AUTOSAR code with a few runnables already implemented. This file is intended to interface with the DaVinci Configurator. When the DaVinci Configurator is configured to output runnables at this directory, the DaVinci Configurator might add new empty runnables or change the names of the existing ones, but preserve the existing implementation. The empty runnables still need to be implemented manually. This file is intended to help reduce manual implementation effort required to integrate DDS into your AUTOSAR application.

To integrate into your AUTOSAR application:

1. Include the generated DDS header files in your adapter layer.
2. Call the adapter APIs to initialize DDS participants and allocate publishers and subscribers.
3. Link the DDS and adapter object files into your application binary.
4. Configure AUTOSAR memory and timing constraints using the generated configuration files.

---

**Note:** Any required manual modifications to the file are marked with `TODO` comments.

---

#### Examples of generated code

When you run MAG with the DDS System XML configuration and deployment options, the tool generates a complete example of a deployable application structure. The generated code is organized by deployment, allowing you to maintain separate code artifacts for different ECUs and deployment scenarios.

This section shows the structure and content of the output artifacts generated for the example mentioned above, including representative samples from header files, source files, and generated configuration.

#### Deployment-specific header file and source file

Generated header files contain deployment-targeted macros and configuration that reflect the selected deployment's topology, participants, and QoS settings.

Generated source files populate initialization arrays for discovery transport configuration, peer discovery lists, and QoS profiles derived from your XML definition. This code is generated specifically for the selected deployment. These files have the same structure as when MAG is running in non-deployment mode. Please refer to `Appgen.h` and `Appgen.c` for more detail.

#### Directory structure

When you run MAG with the `-deployment` and `-applicationType AUTOSAR_CDD` options, the generated code is organized hierarchically by deployment scenario and deployment name. This structure enables maintaining separate code artifacts for different ECUs without file name conflicts:

```
MyDeploymentLib/
  MyDeploymentScenario/
      MyDeployment1/
          AutosarApp/
              MCU_DDS_CDD/
                  adaptation /
                      dds_cdd_adapter.h
                      dds_cdd_adapter.c
                  autosar_gen /
                      Autosar runnables DdsCdd.c
                  autosar_model /
                      DdsCddType.arxml
                  dds_gen (``rtiarcgen``) /
                      *conversions.h
                      *conversions.c
                      *types.arxml
                      *types.h
                      *types.c
                      *typesPlugin.h
                      *typesPlugin.c
                      *typesSupport.h
                      *typesSupport.c
                  dds_impl /
                      dds_impl.h
                      dds_impl.c
                      *Appgen.h
                      *Appgen.c
              MCU_DDS_CDD_EXAMPLE_ASWC /
                  autosar_model /
                      application.arxml
```

The key points of this structure are:

- **Deployment Isolation**: Each deployment generates independent code in its own directory subtree, preventing file name collisions in multi-deployment builds.

- **DDS Layer**: The `dds_gen/` directory contains portable DDS type support code (serialization plugins, data types, and RTE/DDS type conversions) generated by `rtiarcgen` that can be shared if multiple deployments use the same topics. The `dds_impl` directory contains DDS related implementations such as read and write functions along with DDS entities creation.
- **Adapter Layer**: The `adaptation/` directory contains deployment-specific bridge code connecting DDS implementation to your AUTOSAR application code.
- **AUTOSAR Configuration**: The `autosar_gen` contain generated C header/source files for runnables implementation, and `autosar_model/` contains AUTOSAR XML (ARXML) configuration files for toolchain integration.

---

**Note:** The generated ARXML files, such as `DdsCddType.arxml`, `application.arxml`, and `*_types.arxml` are validated against the AUTOSAR schema R4.0.3. However, you may need to customize these files to fit specific requirements. Refer to *Custom AUTOSAR Templates* for more details.

---

### DDS layer

The most notable file from this layer is the `dds_impl.c` that provides functions to create and enable DDS entities.

```
int DdsImpl_CreateEntities()
{
    DDS_DomainParticipantFactory *factory = NULL;

    RT_Registry_T *registry = NULL;
    struct APPGEN_FactoryProperty model_xml = APPGEN_FactoryProperty_INITIALIZER;
    char *participant_name = NULL;

#if (OSAPI_ENABLE_LOG == 1 && OSAPI_ENABLE_TRACE == 1)
    /* Configure RTI DDS Micro logging */
    struct OSAPI_LogProperty property;
    OSAPI_Log_get_property(&property);
    property.write_buffer = my_log_write_function;
    OSAPI_Log_set_verbosity(OSAPI_LOG_VERBOSITY_ERROR);
    if (!OSAPI_Log_set_property(&property)) {
        printf("Failed to set log property\n");
    }
#endif

    OSAPI_Heap_allocate_buffer(
            (char **)&application,
            sizeof(struct Application),
            OSAPI_ALIGNMENT_DEFAULT);
    if (application == NULL)
    {
        printf("failed to allocate application\n");
        return -1;
```

(continues on next page)

(continued from previous page)

```
    }

    /* Retrieve Domain Participant Factory and registry */
    factory = DDS_DomainParticipantFactory_get_instance();
    registry = DDS_DomainParticipantFactory_get_registry(factory);

    /* Register Appgen */
    model_xml._model = APPGEN_get_library_seq();
    if (!APPGEN_Factory_register(registry, &model_xml))
    {
        printf("DdsImpl_CreateEntities: failed to register Application Generation\n");
        return -1;
    }

    /* Create participant from XML profile */
    participant_name = "AppLibrary::MCU_DDS_CDD";
    application->MCU_DDS_CDD =
    DDS_DomainParticipantFactory_create_participant_from_config(
            factory,
            participant_name);

    if (application->MCU_DDS_CDD == NULL)
    {
        printf("DdsImpl_CreateEntities: failed to create participant %s\n",
        participant_name);
        return -1;
    }

    return 0;
}

int DdsImpl_EnableEntities(void)
{
    DDS_ReturnCode_t retcode;
    if (application == NULL) {
        printf("DdsImpl_EnableEntities: application is NULL\n");
        return -1;
    }
    retcode = DDS_Entity_enable(DDS_DomainParticipant_as_entity(application->MCU_DDS_
↪CDD));
    if (retcode != DDS_RETCODE_OK) {
        printf("DdsImpl_EnableEntities: Failed to enable participant "
        "AppLibrary::MCU_DDS_CDD\n");
        return -1;
    }

    return 0;
}
```

`APPGEN_get_library_seq` is defined in `hello_worldAppgen.c` to get DDS models that represent the DDS system specified in the system definition XML. `APPGEN_Factory_register` creates DDS entities required by the system definition XML with specified QoS.

dds_impl.c also provides DDS write and read functions:

```
int DdsImpl_HelloWorldTopic_WriteSample(const dds_HelloWorld* dds_sample)
{
    DDS_ReturnCode_t retcode;
    DDS_DataWriter *datawriter_untyped = NULL;
    static dds_HelloWorldDataWriter *datawriter = NULL;
    const char *datawriter_name = "MCU_DDS_CDD_Pub::MCU_DDS_CDD_HelloWorld_DW";

    if (dds_sample == NULL)
    {
        return -1;
    }

    datawriter_untyped = DDS_DomainParticipant_lookup_datawriter_by_name(
            application->MCU_DDS_CDD,
            datawriter_name);
    if (datawriter_untyped == NULL)
    {
        printf("DdsImpl_HelloWorldTopic_WriteSample: datawriter_untyped == NULL\n");
        return -1;
    }
    datawriter = dds_HelloWorldDataWriter_narrow(datawriter_untyped);
    if (datawriter == NULL)
    {
        printf("DdsImpl_HelloWorldTopic_WriteSample: datawriter == NULL\n");
        return -1;
    }

    /* Write sample to DDS DataWriter */
    retcode = dds_HelloWorldDataWriter_write(
            datawriter,
            dds_sample,
            &DDS_HANDLE_NIL);

    if (retcode != DDS_RETCODE_OK)
    {
        printf("DdsImpl_HelloWorldTopic_WriteSample: Failed to write sample, retcode(%d)\
↪n", retcode);
        return -1;
    }

    return 0;
}

int DdsImpl_HelloWorldTopic_TakeNextSample(dds_HelloWorld* dds_sample, int* is_valid)
{
    DDS_ReturnCode_t retcode;
    struct DDS_SampleInfo sample_info = DDS_SampleInfo_INITIALIZER;
    DDS_DataReader *datareader_untyped = NULL;
    static dds_HelloWorldDataReader *datareader = NULL;
    const char *datareader_name = "MCU_DDS_CDD_Sub::MCU_DDS_CDD_HelloWorld_DR";
```

(continues on next page)

(continued from previous page)

```
    if (dds_sample == NULL || is_valid == NULL)
    {
        return -1;
    }

    datareader_untyped = DDS_DomainParticipant_lookup_datareader_by_name(
            application->MCU_DDS_CDD,
            datareader_name);
    if (datareader_untyped == NULL)
    {
        printf("DdsImpl_HelloWorldTopic_TakeNextSample: datareader_untyped == NULL\n");
        return -1;
    }

    datareader = dds_HelloWorldDataReader_narrow(datareader_untyped);
    if (datareader == NULL)
    {
        printf("DdsImpl_HelloWorldTopic_TakeNextSample: datareader == NULL\n");
        return -1;
    }

    *is_valid = 0;

    /* Take next sample from DataReader */
    retcode = dds_HelloWorldDataReader_take_next_sample(
            datareader,
            dds_sample,
            &sample_info);

    if (retcode == DDS_RETCODE_NO_DATA)
    {
        /* No data available - not an error */
        return 0;
    }

    if (retcode != DDS_RETCODE_OK)
    {
        printf("DdsImpl_HelloWorldTopic_TakeNextSample: Failed to take sample, retcode(
↪%d)\n", retcode);
        return -1;
    }

    /* Check if the sample contains valid data */
    if (sample_info.valid_data)
    {
        *is_valid = 1;
    }

    return 0;
}
```

#### Adapter layer

Deployment-specific bridge code connecting DDS implementation to your AUTOSAR application code. The most notable thing about this layer is that it calls DDS functions from the DDS layer and sets the AUTOSAR system properties. These properties must be specified in the system definition XML:

```
<deployment_library name="MyDeploymentLib">
    <deployment_scenario name="MyDeploymentScenario">
        <deployment name="MyDeployment1">
            …
            <configuration>
                <autosar>
                    <heap>
                        <element>
                            <size>240</size>
                            <start_address>0x60000100</start_address>
                            <name>Heap_area_1</name>
                        </element>
                        <element>
                            <size>96</size>
                            <start_address>0x50000100</start_address>
                            <name>Heap_area_2</name>
                        </element>
                        <element>
                            <size>96</size>
                            <start_address>0x40000100</start_address>
                            <name>Heap_area_3</name>
                        </element>
                        <element>
                            <size>96</size>
                            <start_address>0x30000100</start_address>
                            <name>Heap_area_4</name>
                        </element>
                        <element>
                            <size>96</size>
                            <start_address>0x10000100</start_address>
                            <name>Heap_area_5</name>
                        </element>
                    </heap>
                    <mutex_resource_id>MutexResource1</mutex_resource_id>
                    <max_local_addr_id>2</max_local_addr_id>
                    <send_local_addr_id>2</send_local_addr_id>
                </autosar>
            </configuration>
        </deployment>
    </deployment_scenario>
</deployment_library>
```

The generated output from the above XML is as follows:

```
#define DDSCDD_NUMBER_OF_HEAP_AREAS    5


#define DDSCDD_HEAP_AREA_1_SIZE (240 * 1024) - 0x200
#define DDSCDD_HEAP_AREA_2_SIZE (96 * 1024) - 0x200
#define DDSCDD_HEAP_AREA_3_SIZE (96 * 1024) - 0x200
#define DDSCDD_HEAP_AREA_4_SIZE (96 * 1024) - 0x200
#define DDSCDD_HEAP_AREA_5_SIZE (96 * 1024) - 0x200

    static const uint32 heap_area_size[DDSCDD_NUMBER_OF_HEAP_AREAS] =
{
            DDSCDD_HEAP_AREA_1_SIZE,
            DDSCDD_HEAP_AREA_2_SIZE,
            DDSCDD_HEAP_AREA_3_SIZE,
            DDSCDD_HEAP_AREA_4_SIZE,
            DDSCDD_HEAP_AREA_5_SIZE
};

static char heap_area1[DDSCDD_HEAP_AREA_1_SIZE] __at(0x60000100);
static char heap_area2[DDSCDD_HEAP_AREA_2_SIZE] __at(0x50000100);
static char heap_area3[DDSCDD_HEAP_AREA_3_SIZE] __at(0x40000100);
static char heap_area4[DDSCDD_HEAP_AREA_4_SIZE] __at(0x30000100);
static char heap_area5[DDSCDD_HEAP_AREA_5_SIZE] __at(0x10000100);

static char* const heap_area[DDSCDD_NUMBER_OF_HEAP_AREAS] =
{
        heap_area1,
        heap_area2,
        heap_area3,
        heap_area4,
        heap_area5
};

static int SetSystemProperties(void)
{
    struct OSAPI_SystemAutosar system_property;

    if (!OSPSL_AutosarSystem_get_property(&system_property))
    {
        printf("failed to get system properties\n");
        return -1;
    }

    /* Task OSAPI_SystemAutosar_timer_task is configured to run every 10 ms */
    system_property.psl_property.timer_resolution_ms = 10;

    /* Configure static memory heap areas */
    system_property.psl_property.number_of_heap_areas = DDSCDD_NUMBER_OF_HEAP_AREAS;
    system_property.psl_property.heap_area_size = heap_area_size;
    system_property.psl_property.heap_area = (const char **)heap_area;
    system_property.psl_property.enable_thread_safe_heap =
    RTI_FALSE; /* TODO: Default is not thread safe, change to true if using multithread␣
↪*/
```

(continues on next page)

(continued from previous page)

```
    /* Connext DDS Micro will use Resources as synchronization method */
    system_property.psl_property.sync_type = OSAPI_AUTOSAR_SYNCKIND_RESOURCES;
    system_property.psl_property.mutex_resource_id = MutexResource1;

    /* AUTOSAR synchronization configuration - Resources only */
    system_property.psl_property.semaphore_max_count = 0;
    system_property.psl_property.first_give_event = 0;
    system_property.psl_property.first_timeout_event = 0;
    system_property.psl_property.first_alarm = 0;

    /* Enable AUTOSAR TcpIp Socket Owner integration */
    system_property.psl_property.use_socket_owner = TRUE;

    /* Configure socket limits based on multicast support */
#if defined(NETIO_CONFIG_ENABLE_MULTICAST) && (NETIO_CONFIG_ENABLE_MULTICAST)
    system_property.psl_property.max_receive_sockets = 4;  /* With multicast */
#else
    system_property.psl_property.max_receive_sockets = 2;  /* Unicast only */
#endif

    /* Disable internal UDP buffers - use AUTOSAR TcpIp stack buffers */
    system_property.psl_property.number_of_rcv_buffers = 0;
    system_property.psl_property.rcv_buffer_size = 0;

    /* Set AUTOSAR TcpIp integration callbacks */
    system_property.psl_property.get_socket = DdsCdd_GetSocket;
    system_property.psl_property.send_data = NULL;
    system_property.psl_property.max_local_addr_id = 2;

    system_property.psl_property.send_local_addr_id = 2;

    /* Configure UDP thread handling - use synchronous mode for AUTOSAR */
    system_property.psl_property.use_udp_thread = FALSE;
    system_property.psl_property.dds_rxindication =
    DdsCddRxIndication; /* TODO: Default name for dds rx indication, change if using␣
↪custom callback */

    if (!OSPSL_AutosarSystem_set_property(&system_property))
    {
        printf("failed to set system properties\n");
        return -1;
    }

    return 0;
}

/* TcpIp_[SocketOwnerName]GetSocket - TODO: update if using different SocketOwner name */
#define DDSCDD_SOCKET_OWNER_GET_SOCKET   TcpIp_DdsCddGetSocket
```

For more information on how to populate these properties, refer to *OSAPI_PortProperty reference*.

**Note:** Properties with `TODO` comments require close attention from the integrator to check and make sure they have the correct values based on the peripheral AUTOSAR environment.

### Custom AUTOSAR Templates

MAG supports custom AUTOSAR templates to allow users to customize the generated files' contents. To use custom templates, create a directory with the same structure as the default templates in `RTIMEHOME/rtiddsmag/resource/templates/autosar`:

```
autosar
   application
       autosar_model
           application.arxml.vm
           application_macro.arxml.vm
   arxml.xml.vm
   CDD
       adaptation
           dds_cdd_adapter.c.vm
           dds_cdd_adapter.h.vm
           dds_cdd_adapter_macro.c.vm
           dds_cdd_adapter_macro.h.vm
       autosar_gen
           DdsCdd.c.vm
           DdsCdd_macros.c.vm
       dds_impl
            dds_impl.c.vm
            dds_impl.h.vm
            dds_impl_macro.c.vm
            dds_impl_macro.h.vm
```

Then, run `rtiddsmag` with the `-autosarTemplatePath <Path>` option to point to your custom template directory.

The `<Path>` can be an absolute path or a path relative to the current working directory that is pointing to the directory containing application, arxml.xml.vm, and/or CDD folders.

If the custom template directory is missing any of the required template files, MAG will fall back to the default template for those files.

**Note:** Overriding the default template should be done with caution. Please contact RTI support for more information.

For example, you have the following custom template directory:

```
my_autosar_templates
   application
       autosar_model
           application.arxml.vm
```

```
rtiddsmag -inputXml hello_world.xml -deployment MyDeployment1 -applicationType AUTOSAR_
↪CDD -language C -replace -autosarTemplatePath ./my_autosar_templates
```

While `my_autosar_templates` contains only the `application/autosar_model/application.arxml.vm` template, MAG will use the custom template for generating `application.arxml` and use the default templates for all other generated files.

This allows you to customize only the files you need, while still benefiting from the default generation for the rest of the files and keeping the modifications separate from the default templates.

#### Limitations and considerations

Take note of the following known limitations and design considerations when generating AUTOSAR code with MAG:

- **Deployment Isolation**: Each deployment generates independent code. If multiple deployments use the same topics and types, you may observe code duplication in the generated DDS layer (serialization/deserialization plugins). This is intentional and allows each deployment to be compiled and deployed autonomously.
- **AUTOSAR Resource Configuration**: The `<autosar>` configuration block in deployment sections supports memory heap definitions and resource range specification for AUTOSAR integration. However, advanced AUTOSAR configuration (timing, scheduling, runnable mapping) must be completed manually in your AUTOSAR toolchain after code generation.
- **XML Schema Validation**: The DDS System XML schema is located at `resource/schema/dds-xml_system_definitions.xsd`. Ensure your XML editor or IDE is configured to validate against this schema for development and testing. Validation errors during schema checking do not prevent MAG execution but may indicate configuration issues that impact deployment-aware code generation.
- **XML Discrepancies with |core_pro|**: There are discrepancies between what the *Connext Professional* XSD supports and what *Connext Micro* supports. In the *Connext Professional* XSD, `transportBuiltinQosPolicy:udpv4` allows socket buffer size fields to be specified as strings. If one of the Pro socket buffer size enum constants is provided instead of a numeric value, MAG cannot verify that `max_message_size` is consistent and will emit a warning.
- **Custom AUTOSAR Templates**: When using the `-autosarTemplatePath <Path>` parameter, custom templates must have the same directory structure and names as the default templates `RTIMEHOME/rtiddsmag/resource/templates/autosar`, only the content of the template files can be different. Although MAG will fall back to the default template if the custom template is missing.

### 3.17.6 Generating native DDS applications with deployment-aware code generation

> **Attention:** This is an experimental feature in *Connext Micro* 4.3.0. It is not guaranteed to be consistent or supported and should not be used in production. C++ is not yet supported for this workflow.
>
> Refer to *Experimental Features* for more information.

This section describes how to use deployment-aware code generation to create native DDS applications for embedded or standalone systems. Unlike AUTOSAR applications, native applications have direct control over the DDS middleware and can be deployed to various platforms without the AUTOSAR framework. The process involves:

1. Defining a complete system in DDS system XML format.
2. Running `rtiddsmag` with `-deployment` and `-applicationType micro4` selectors to generate deployment-specific code.

   > **Attention:** To generate native application code, `-deployment` and `-applicationType` *must be specified together*. The `-applicationType micro4` option generates code optimized for *Connext Micro* runtime deployment.

3. Generating type support files with `rtiddsgen`.
4. Setting up the runtime environment variables (`RTIMEHOME` and `RTIME_TARGET_NAME`) to point to your *Connext Micro* installation.
5. Building the application using CMake, which automatically resolves *Connext Micro* dependencies and generates the native executable.

We'll go over each of these steps in more detail below.

#### Define your system in DDS system XML

The DDS System XML format allows you to define the complete topology of your distributed system, including *Topics*, *DomainParticipants*, nodes, and deployment configurations. For native applications, follow the same XML structure as AUTOSAR applications (see *Define your system in DDS system XML* for detailed information on XML schema and structure).

The key difference for native applications is that the deployment configuration should target a native platform, rather than an AUTOSAR ECU. For native applications, the configuration tag is not required and does not apply.

---

**Note:** Each deployment configuration generates code for a single application. The schema supports multiple applications per deployment, but MAG's current implementation generates code for

only one application. Undefined behavior may occur if multiple applications are defined under the same deployment.

The following example demonstrates a snippet of a DDS System XML file with a deployment configuration for a native application:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="resource/schema/dds-xml_system_definitions.xsd">

    ...

    <!-- Deployment configuration -->
    <deployment_library name="MyDeploymentLib">
        <deployment_scenario name="MyDeploymentScenario">
            <deployment name="MyDeployment1">
                <node node_ref="MyNodeLib::MyNode1" />
                <applications>
                    <application name="NativeApp" application_ref=
↪"MyApplicationLib::MyApplication" />
                </applications>
                <!-- <configuration> is not required for native applications -->
            </deployment>
        </deployment_scenario>
    </deployment_library>

</dds>
```

### Run MAG with deployment and application type selectors

After creating your DDS System XML file, use `rtiddsmag` to generate deployment-aware code for native application targets. Run the following command with your XML and deployment name:

Linux

```
$RTIMEHOME/bin/rtiddsmag -language C <dds_system>.xml -d output -replace \\
    -deployment MyDeployment1 -applicationType micro4 -verbosity 3
```

macOS

```
$RTIMEHOME/bin/rtiddsmag -language C dds_system.xml -d output -replace \\
    -deployment MyDeployment1 -applicationType micro4 -verbosity 3
```

Windows

```
%RTIMEHOME%\bin\rtiddsmag.bat -language C dds_system.xml -d output -replace ^
    -deployment MyDeployment1 -applicationType micro4 -verbosity 3
```

The `-deployment` option selects the deployment scenario from your XML, and the `-applicationType micro4` option generates code optimized for *Connext Micro* runtime execution. Both options are required when using the deployment-aware workflow for native applications. `rtiddsmag` will generate the following:

- Deployment-specific source code organized by deployment name.
- `*Appgen.c` and `*Appgen.h` files containing the DDS entity initialization code for the selected deployment.
- Generated C header and source files containing DDS entity definitions, type support code, and application configuration.
- A `CMakeLists.txt` file to build the application with proper *Connext Micro* library linkage.
- A `main.c` file that demonstrates the basic application lifecycle.
- Optional type-specific implementation files for handling data reads and writes.

#### Generate type support files with rtiddsgen

Next, generate type support files for your data types using `rtiddsgen`, as shown below:

> **Attention:** If you use an XML file (such as `dds_system.xml`) as input for `rtiddsgen`, you must call it at a specific directory to validate the XML file against the schema.
>
> Since dds_system.xml specifies the schema location as `resource/schema/dds-xml_system_definitions.xsd`, which refers to the `$RTIMEHOME/rtiddsmag/resource/schema/dds-xml_system_definitions.xsd`, the current working directory must be set to `$RTIMEHOME/rtiddsmag` when calling `rtiddsgen` with `dds_system.xml` as input.

Linux

```
cd $RTIMEHOME/rtiddsmag

$RTIMEHOME/bin/rtiddsgen -create typefiles -micro -language C dds_system.xml -d <path_to_
↪output>/<deployment_library_name>/<scenario_name>/<deployment_name>/<application_name>
```

macOS

```
cd $RTIMEHOME/rtiddsmag

$RTIMEHOME/bin/rtiddsgen -create typefiles -micro -language C dds_system.xml -d <path_to_
↪output>/<deployment_library_name>/<scenario_name>/<deployment_name>/<application_name>
```

Windows

```
cd %RTIMEHOME%\rtiddsmag

%RTIMEHOME%\bin\rtiddsgen.bat -create typefiles -micro -language C dds_system.xml␣
↪-d <path_to_output>\<deployment_library_name>\<scenario_name>\<deployment_name>\
↪<application_name>\
```

### Understanding the generated files

When you run the above commands, `rtiddsmag` and `rtiddsgen` generate code with the following directory structure:

```
output/
  MyDeploymentLib (<deployment_library_name>) /
      MyDeploymentScenario (<scenario_name>) /
          MyDeployment1 (<deployment_name>) /
              NativeApp (<application_name>) /
                  CMakeLists.txt
                  main.c
                  NativeApp.h (<application name>.h)
                  NativeApp.c (<application name>.c)
                  dds_system.h (<type xml filename>.h)
                  dds_system.c (<type xml filename>.c)
                  dds_systemPlugin.h (<type xml filename>Plugin.h)
                  dds_systemPlugin.c (<type xml filename>Plugin.c)
                  dds_systemSupport.h (<type xml filename>Support.h)
                  dds_systemSupport.c (<type xml filename>Support.c)
                  dds_systemAppgen.h (<type xml filename>Appgen.h)
                  dds_systemAppgen.c (<type xml filename>Appgen.c)
                  build_test/
                    (build output directory)
```

The files generated by `rtiddsmag` for native applications serve specific purposes in the application lifecycle:

- **Application Header and Implementation**

  `NativeApp.h` and `NativeApp.c`: The main application interface that abstracts DDS entity creation and lifecycle management. This file contains the `Application` structure, initialization functions (`DDS_Configure()`, `DDS_Init()`, `DDS_Run()`), and wrapper functions for publishing and subscribing to topics.

- **Type Support Files**

  These are generated by `rtiddsgen` based on the data types defined in your XML configuration:

  - `dds_system.h`: Contains the C structure definitions for all data types defined in your XML configuration.
  - `dds_systemPlugin.h` and `dds_systemPlugin.c`: Provide type plugin factories that enable serialization and deserialization of your data types for DDS transport.
  - `dds_systemSupport.h` and `dds_systemSupport.c`: Contain utility functions for creating, managing, and operating on your data types.

- **Application Generation Files**

  - `dds_systemAppgen.h` and `dds_systemAppgen.c`: Generated by `rtiddsmag` to contain the DDS entity configurations and initialization code. This file implements the `DDS_Configure()` function that creates all DomainParticipants, Publishers, Subscribers, DataWriters, and DataReaders as specified in your XML deployment.

- **Build Configuration**

  `CMakeLists.txt`: Defines the build process for your application. It:

  - Requires `RTIMEHOME` and `RTIME_TARGET_NAME` environment variables;
  - Links against the appropriate *Connext Micro* libraries for your target platform;
  - Compiles all source files;
  - Produces a native executable in the `objs/<TARGET_NAME>/` directory.

- **Entry Point**

  `main.c`: A simple main function that demonstrates the basic application lifecycle: configure DDS entities, initialize the DDS layer, and run the application event loop.

#### Building and running native applications

Before building, ensure that the following environment variables are set:

- `RTIMEHOME`: Path to your *Connext Micro* installation directory.
- `RTIME_TARGET_NAME`: The target platform identifier (e.g., `x64Linux4gcc7.3.0` for 64-bit Linux with GCC 7.3.0).

Then run the following command:

Linux/macOS

```
export RTIMEHOME=/path/to/rti_connext_dds_micro-4.3.0
export RTIME_TARGET_NAME=x64Linux4gcc7.3.0
cd output/<deployment_library_name>/<scenario_name>/<deployment_name>/<application_name>
cmake -S . -B build_test
cmake --build build_test -j
```

Windows

```
set RTIMEHOME=C:\path\to\rti_connext_dds_micro-4.3.0
set RTIME_TARGET_NAME=x64Win64VS2015
cd output\<deployment_library_name>\<scenario_name>\<deployment_name>\<application_name>
cmake -S . -B build_test
cmake --build build_test -j
```

After a successful build, the native executable will be located in:

```
objs/<RTIME_TARGET_NAME>/<application_name>
```

You can run the application directly from the command line:

Linux/macOS

```
./objs/x64Linux4gcc7.3.0/<application_name>
```

Windows

```
objs\x64Win64VS2015\<application_name>.exe
```

#### Customizing native applications

Due to MAG's limited ability to infer type definitions, the generated code does populate the sample with default values, but does not read actual data fields for your types. It only prints out when a sample is published or received, without accessing the relevant data fields.

You can customize your application's behavior with some hooks in the generated `<application_name>.c` file:

- **Topic Write Functions**: For each *DataWriter* in your configuration, a function like `<topic>_write(const <type> *data)` is generated. Modify this function to populate and publish data to the corresponding topic.
- **Topic Read Callbacks**: For each *DataReader* in your configuration, a function like `<topic>_read(const <type> *data)` is generated. This function is called whenever data is received on the corresponding topic. Implement this to read the fields of the incoming data.
- **Main Application Loop**: Modify the `DDS_Run()` function to implement your application's event loop, including periodic publishing, responding to callbacks, or application-specific business logic.

#### Troubleshooting

Aside from *Errors caused by invalid configurations*, here are some solutions to possible issues with generating AUTOSAR applications:

- **Schema validation errors**

  Ensure your XML file includes the correct schema location: `xsi:noNamespaceSchemaLocation="resource/schema/dds-xml_system_definitions.xsd"`.

- **Generated code will not compile**

  Check that all referenced types (via the `type_ref` and `base_name` attributes) are defined in the same XML file or imported via type registration.

- **rtiarcgen reports template or tool errors**

  Run `rtiarcgen -selfTest` and verify your environment and tool installation before generation. If using type support generation in other flows, verify that `RTIDDSGEN_PATH` is configured.

- **Import unresolved references in AUTOSAR toolchain**

  Import order matters when using the Vector DaVinci toolchain. Import the `rtiarcgen` types ARXML first, then import the MAG-generated AUTOSAR configuration ARXML for each deployment.

### 3.17.7 Errors caused by invalid configurations

This section explains the different results thrown by MAG if it receives invalid configuration files.

- **Invalid XML content**

  MAG will fail to validate the configuration file if it contains invalid content, such as elements/attributes that don't exist in the schema or values that aren't supported by any of the existing types. For example:

```
<dds>
    ...
    <!-- Participant Library -->
    <domain_participant_library name="FeatureTestLibrary">
        <domain_participant name="01_EmptyDomainParticipant"
        domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <invalid_tag></invalid_tag>
        </domain_participant>
    </domain_participant_library>
    ...
</dds>
```

```
07:41:48.334 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Error.xml
07:41:49.827 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to parse inp
ut file : /home/test/Error.xml
07:41:49.837 [main] ERROR com.rti.micro.appgen.MicroAppGen - cvc-complex-type.2.
4.a: Invalid content was found starting with element 'invalid_tag'. One of '{dat
a_writer, publisher_qos}' is expected.
07:41:49.837 [main] INFO  com.rti.micro.appgen.MicroAppGen - Exiting.
```

- **Unsupported elements**

  MAG will throw a warning for any elements that are not supported by *Connext Micro*. Unsupported elements will be ignored, such as the user_data in the following:

```
<dds>
    ...
    <!-- Participant Library -->
    <domain_participant_library name="FeatureTestLibrary">
        <domain_participant name="01_EmptyDomainParticipant"
        domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <domain_participant_qos>
                <!-- user_data is not supported by Micro -->
                <user_data/>
            </domain_participant_qos>
        </domain_participant>
    </domain_participant_library>
</dds>
```

```
07:39:52.643 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Warning.xml
07:39:53.439 [main] WARN  com.rti.micro.appgen.utils.ConverterUtils - userData i
s not supported by Micro, the tool will ignore its value.
file=/home/test/Warning.xml, lineNumber=90, columnNumber=38
```

- **Unsupported values**

MAG will throw an error if it finds a value that is not supported by *Connext Micro*.

```
<dds>
    ...
    <!-- Participant Library -->
    <domain_participant_library name="FeatureTestLibrary">
        <domain_participant name="01_EmptyDomainParticipant"
        domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <publisher name ="test">
                <data_writer topic_ref="HelloWorldTopic1" name="testW">
                    <datawriter_qos>
                        <durability>
                            <!-- transient is not supported by Micro -->
                            <kind>TRANSIENT_DURABILITY_QOS</kind>
                        </durability>
                    </datawriter_qos>
                </data_writer>
            </publisher>
        </domain_participant>
    </domain_participant_library>
</dds>
```

```
07:39:01.248 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Error.xml
07:39:02.069 [main] ERROR com.rti.micro.appgen.utils.ConverterUtils - TRANSIENT_
DURABILITY_QOS is not supported by Micro, only VOLATILE and TRANSIENT_LOCAL are
valid values for the durability kind field.
file=/home/test/Error.xml, lineNumber=35, columnNumber=37
07:39:02.072 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to add input
 file information into the model.
07:39:02.074 [main] INFO  com.rti.micro.appgen.MicroAppGen - Exiting.
```

MAG will throw an error if the QoS values are not consistent with values supported in *Connext Micro*. For example, the following XML contains a deadline period that is too large.

```
<dds>
    ...
    <!-- Participant Library -->
    <domain_participant_library name="FeatureTestLibrary">
        <domain_participant name="01_EmptyDomainParticipant"
        domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <publisher name ="test">
                <data_writer topic_ref="HelloWorldTopic1" name="testW">
                    <datawriter_qos>
                        <deadline>
                            <!-- this deadline exceeds the maximum -->
                            <period>
                                <sec>123213123</sec>
                                <nanosec>12</nanosec>
                            </period>
                        </deadline>
                    </datawriter_qos>
                </data_writer>
            </publisher>
```

(continues on next page)

(continued from previous page)

```
        </domain_participant>
    </domain_participant_library>
</dds>
```

```
07:43:26.805 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Error.xml
07:43:27.619 [main] ERROR com.rti.micro.appgen.utils.ConverterUtils - The durati
on of deadline.period=3.907062500000000038052 y exceeded the maximum range [1 ns,
 1 year]
file=/home/test/Error.xml, lineNumber=35, columnNumber=11
07:43:27.620 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to add input
 file information into the model.
07:43:27.620 [main] INFO  com.rti.micro.appgen.MicroAppGen - Exiting.
```

MAG will throw an error if the `dds.xtypes.compliance_mask` property uses a different value than 0x00000008.

```
file=/tmp/Error.xml, lineNumber=184, columnNumber=38
09:29:36.451 [main] ERROR com.rti.micro.appgen.utils.ConverterUtils - 0x00000009 is not a valid value for dds.xtypes.compliance_mask, only 0x00000008 is supported.
file=/tmp/Error.xml, lineNumber=305, columnNumber=27
09:29:36.451 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to calculate the system model.
java.lang.Exception: Invalid values found while creating the Model.
```

- **Unsupported QoS**

  Not all the QoS policies supported by *Connext Micro* can be configured in XML.

  – QoS settings related to UDP transformation cannot be configured in XML. See the *UDP Transport* section for more information on UDP transformation.

  – MAG does not support any PROPERTY QoS policy properties except the `dds.xtypes.compliance_mask` property.

- **Unknown deployment name**

  MAG will throw an error if the `-deployment` option is specified with a name that is not defined in your XML file.

  Verify that the deployment name specified with `-deployment` exactly matches a deployment defined in your XML file's `<deployment_library>/<deployment_scenario>/<deployment>` hierarchy.

- **Application type AUTOSAR_CDD not supported without -deployment**

  The `-applicationType AUTOSAR_CDD` option requires the `-deployment` option. Specify both to enable AUTOSAR CDD code generation.

## 3.18 QoS Mapping in XML

QOS definitions in XML are based on the "standard" mapping of QOS policy names and members as found in the *Connext Micro* documentation, plus the following additional members that are parsed by RTI MAG:

### 3.18.1 Domain Participant Qos QoS Mapping

The following mapping documents the currently implemented MAG behavior for `domain_participant_qos` conversion that are additional to what Micro supported.

General behavior:

- Tags present in Pro XML but not supported by Micro are ignored by MAG.
- MAG emits a warning when unsupported fields are encountered.
- Some fields are accepted by parser/validation, but only a subset is emitted into generated Micro configuration.

#### discovery_config

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `builtin_discovery_plugins` | Selects discovery kind (`DPDE` or `DPSE`) | Discovery component type |
| `participant_liveliness_lease_duration` | Consumed | DPDE/DPSE discovery property |
| `participant_liveliness_assert_period` | Consumed | DPDE/DPSE discovery property |
| `initial_participant_announcements` | Consumed | DPDE/DPSE discovery property |
| `max_initial_participant_announcements` or `min_initial_participant_announcements` | Consumed (announcement period value is set to `1`) | DPDE/DPSE discovery property |
| `participant_message_reader_reliability_kind` | Consumed | DPDE/DPSE discovery property |
| `max_participant_locators` | Consumed | DPDE/DPSE discovery property |
| `max_locators_per_discovered_participant` | Consumed | DPDE/DPSE discovery property |
| `cache_serialized_samples` | Consumed for DPDE | DPDE discovery property |
| `subscription_reader_resource_limits.max_samples` | Consumed for DPDE | `max_samples_per_builtin_endpoint_reader` |
| `subscription_reader_resource_limits.max_samples_per_remote_writer` or `publication_reader_resource_limits.max_samples_per_remote_writer` or `participant_reader_resource_limits.max_samples_per_remote_writer` | Consumed, limited to one Micro writer (maximum, with unlimited handling) | `max_samples_per_remote_builtin_endpoint_writ` |
| `publication_writer.heartbeat_period` | Consumed for DPDE | `builtin_writer_heartbeat_period` |
| `publication_writer.max_heartbeat_retries` | Consumed for DPDE | `builtin_writer_max_heartbeat_retries` |
| `publication_writer.heartbeats_per_max_samples` | Consumed for DPDE | `builtin_writer_heartbeats_per_max_samples` |
| `subscription_reader.nack_period` | Consumed for DPDE | `builtin_endpoint_reader_nack_period` |

#### default_unicast

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `default_unicast.value[].transports` (+ optional `receive_port`) | Converted into participant user-traffic transport/address entries | `user_traffic.enabled_transports` |

**Note:**

- Transport names are normalized to internal/default names.
- If no explicit user traffic is configured, MAG may populate Micro defaults (for example, UDPv4 and INTRA when available).

#### resource_limits

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `*_allocation.max_count` (for participant resource allocations) | Consumed | Corresponding Micro participant resource limits field |
| `*_allocation.initial_count` | Ignored for Micro | None |
| `*_allocation.incremental_count` | Ignored for Micro | None |

**Note:** For `allocation_settings`, `max_count` is the only value used by Micro.

### 3.18.2 Domain Participant Factory Qos QoS Mapping

The following mapping documents the currently implemented MAG behavior for `domain_participant_factory_qos` conversion.

#### Top-level fields

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `entity_factory.autoenable_created_entities` | Consumed | Factory entity factory QoS |
| `resource_limits.max_participants` | Consumed (and auto-raised if needed) | `system_resource_limits.max_participants` |
| `resource_limits.max_components` | Consumed (and auto-raised if needed) | `system_resource_limits.max_components` |
| `base_name` / `base_names` | Used for QoS inheritance/profile resolution | Not emitted as a Micro runtime QoS field |
| `logging` | Ignored for Micro | None |
| `monitoring` | Ignored for Micro | None |

#### Nested resource_limits (systemResourceLimitsQosPolicy)

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `resource_limits.max_participants` | Consumed | `system_resource_limits.max_participants` |
| `resource_limits.max_components` | Consumed | `system_resource_limits.max_components` |

**Note:**

- Unsupported factory QoS fields are warned and ignored by MAG.
- For this nested policy, MAG uses the two fields listed above.

### 3.18.3 Publisher QoS mapping

The following mapping documents the currently implemented MAG behavior for `publisher_qos` conversion.

#### Top-level fields

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `base_name` / `base_names` | Used for QoS inheritance/profile resolution | Not emitted as a Micro runtime QoS field |
| `entity_factory.autoenable_created_entities` | Consumed | `DDS_PublisherQos::entity_factory.autoenable_created_entities` |
| `presentation` | Ignored for Micro — MAG emits a warning | None |
| `partition.name` | Consumed — partition strings emitted as a `const char*` array | `DDS_PublisherQos::partition.name` |
| `group_data.value` | Consumed — octet bytes emitted as a `DDS_Octet` array | `DDS_PublisherQos::group_data.value` |
| `publisher_name.name` | Consumed | `DDS_PublisherQos::publisher_name.name` |
| `asynchronous_publisher` | Ignored for Micro — MAG emits a warning | None |

**Note:**

- Fields not in the supported set cause MAG to emit a warning and reset the field to its default value.
- `partition.name` and `group_data.value` are also referenced in generated remote-publication-data structs (`entities.h`) for DPDE discovery matching.
- `publisher_name.name` is used both in the generated QoS initializer macro and as part of remote publication data struct naming.

### 3.18.4 Subscriber QoS mapping

The following mapping documents the currently implemented MAG behavior for `subscriber_qos` conversion.

#### Top-level fields

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `base_name` / `base_names` | Used for QoS inheritance/profile resolution | Not emitted as a Micro runtime QoS field |
| `entity_factory.autoenable_created_entities` | Consumed | `DDS_SubscriberQos::entity_factory.autoenable_created_entities` |
| `presentation.access_scope` | Consumed | `DDS_SubscriptionBuiltinTopicData::presentati access_scope` |
| `presentation.coherent_access` | Consumed | `DDS_SubscriptionBuiltinTopicData::presentati coherent_access` |
| `presentation.ordered_access` | Consumed | `DDS_SubscriptionBuiltinTopicData::presentati ordered_access` |
| `partition.name` | Consumed | `DDS_SubscriberQos::partition.name` |
| `group_data.value` | Consumed | `DDS_SubscriberQos::group_data.value` |
| `subscriber_name.name` | Consumed | `DDS_SubscriberQos::subscriber_name.name` |

**Note:**

- Fields not in the supported set cause MAG to emit a warning and reset the field to its default value.
- `presentation.*` is consumed when generating remote subscription data (`entities.h`) and is not part of the local `DDS_SubscriberQos` initializer macro.
- `partition.name` and `group_data.value` are used in both the local subscriber QoS initializer macro and generated remote subscription data.

### 3.18.5 Topic QoS mapping

The following mapping documents the currently implemented MAG behavior for `topic_qos` conversion.

### Top-level fields

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `base_name` / `base_names` | Used for QoS inheritance/profile resolution | Not emitted as a Micro runtime QoS field |
| `topic_data.value` | Consumed | `DDS_TopicQos::topic_data.value` and built-in topic-data payload in generated remote endpoint data |
| `durability.*` | Consumed only when `topic_qos` is used as a base for `datawriter_qos` / `datareader_qos` | Inherited into `DDS_DataWriterQos::durability` / `DDS_DataReaderQos::durability` |
| `deadline.period.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::deadline` / `DDS_DataReaderQos::deadline` |
| `latency_budget.duration.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::latency_budget` / `DDS_DataReaderQos::latency_budget` |
| `liveliness.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::liveliness` / `DDS_DataReaderQos::liveliness` |
| `reliability.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::reliability` / `DDS_DataReaderQos::reliability` |
| `destination_order.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::destination_order` / `DDS_DataReaderQos::destination_order` |
| `history.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::history` / `DDS_DataReaderQos::history` |
| `resource_limits.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::resource_limits` / `DDS_DataReaderQos::resource_limits` |
| `ownership.kind` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::ownership` / `DDS_DataReaderQos::ownership` |
| `representation.value` and `representation.compression_settings.*` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::representation` / `DDS_DataReaderQos::representation` |
| `transport_priority.value` | Consumed only through Topic->Writer/Reader QoS inheritance | Inherited into `DDS_DataWriterQos::transport_priority` / `DDS_DataReaderQos::transport_priority` |
| `durability_service.*` | Not consumed by MAG for Topic QoS conversion | None |
| `lifespan.duration.*` | Not consumed by MAG for Topic QoS conversion | None |

**Note:**

- Unlike participant/publisher/subscriber QoS conversion, Topic QoS does not use a dedicated unsupported-field warning set in `ConverterUtils`; unsupported Topic members that are not mapped above remain effectively unused.
- Topic QoS local macro generation (`topicQos` in `qos.h.vm`) emits only management data and `topic_data`.
- Additional Topic fields become effective only when a DataWriter/DataReader inherits from a Topic QoS profile, via `QosUtils.inheritFromTopicQos(...)`.

### 3.18.6 DataReader QoS mapping

The following mapping documents the currently implemented MAG behavior for `datareader_qos` conversion.

### Top-level fields

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `base_name` / `base_names` | Used for QoS inheritance/profile resolution | Not emitted as a Micro runtime QoS field |
| `durability.kind` | Consumed | `DDS_DataReaderQos::durability.kind` |
| `durability.direct_communication`, `durability.storage_settings.*` | Ignored for Micro - MAG emits warnings | None |
| `deadline.period.sec/nanosec` | Consumed | `DDS_DataReaderQos::deadline.period` |
| `liveliness.kind`, `liveliness.lease_duration.*` | Consumed | `DDS_DataReaderQos::liveliness` |
| `reliability.kind`, `reliability.max_blocking_time.*` | Consumed | `DDS_DataReaderQos::reliability` |
| `reliability.acknowledgment_kind`, `reliability.instance_state_consistency_kind`, `reliability.exact_reliability_kind_match` | Ignored for Micro - MAG emits warnings | None |
| `destination_order.kind`, `destination_order.source_timestamp_tolerance.*` | Consumed | `DDS_DataReaderQos::destination_order` |
| `history.kind`, `history.depth` | Consumed | `DDS_DataReaderQos::history` |
| `resource_limits.max_samples`, `resource_limits.max_instances`, `resource_limits.max_samples_per_instance` | Consumed | `DDS_DataReaderQos::resource_limits` |
| `resource_limits.*` (other members) | Ignored for Micro - MAG emits warnings | None |
| `ownership.kind` | Consumed | `DDS_DataReaderQos::ownership.kind` |
| `representation.value.element[]` | Consumed | `DDS_DataReaderQos::representation.value` |
| `representation.compression_settings.*` | Ignored for Micro - MAG emits warnings | None |
| `latency_budget.duration.sec/nanosec` | Consumed | `DDS_DataReaderQos::latency_budget.duration` |
| `user_data.value` | Consumed | `DDS_DataReaderQos::user_data.value` |
| `transport_priority.value` | Consumed | `DDS_DataReaderQos::transport_priority.value` |

**Note:**

- Unsupported fields are detected by `ConverterUtils` against `VALID_DATA_READER_POLICIES`; MAG warns and ignores unsupported values.
- `unicast` and `multicast` are normalized into `microTransports.enabledTransports` by `SystemModelProducer.addEnabledTransports(...)`; the template emits the resulting transport list.
- `content_filter` is configured from the `data_reader` entity-level filter (not a direct `datareader_qos` XML member) and is emitted through `contentFilterQosPolicy`.
- `property` is intentionally not emitted for DataReader QoS in current templates (`propertiesQosPolicy( $null, ...)`).

### 3.18.7 DataWriter QoS mapping

The following mapping documents the currently implemented MAG behavior for `datawriter_qos` conversion.

### Top-level fields

| Pro XML tag | MAG behavior | Micro target |
|---|---|---|
| `base_name` / `base_names` | Used for QoS inheritance/profile resolution | Not emitted as a Micro runtime QoS field |
| `durability.kind` | Consumed | `DDS_DataWriterQos::durability.kind` |
| `durability.direct_communication`, `durability.storage_settings.*` | Ignored for Micro - MAG emits warnings | None |
| `deadline.period.sec/nanosec` | Consumed | `DDS_DataWriterQos::deadline.period` |
| `liveliness.kind`, `liveliness.lease_duration.*` | Consumed | `DDS_DataWriterQos::liveliness` |
| `reliability.kind`, `reliability.max_blocking_time.*` | Consumed | `DDS_DataWriterQos::reliability` |
| `reliability.acknowledgment_kind`, `reliability.instance_state_consistency_kind`, `reliability.exact_reliability_kind_match` | Ignored for Micro - MAG emits warnings | None |
| `destination_order.kind`, `destination_order.source_timestamp_tolerance.*` | Consumed | `DDS_DataWriterQos::destination_order` |
| `history.kind`, `history.depth` | Consumed | `DDS_DataWriterQos::history` |
| `resource_limits.max_samples`, `resource_limits.max_instances`, `resource_limits.max_samples_per_instance` | Consumed | `DDS_DataWriterQos::resource_limits` |
| `resource_limits.*` (other members) | Ignored for Micro - MAG emits warnings | None |
| `ownership.kind` | Consumed | `DDS_DataWriterQos::ownership.kind` |
| `ownership_strength.value` | Consumed | `DDS_DataWriterQos::ownership_strength.value` |
| `representation.value.element[]` | Consumed | `DDS_DataWriterQos::representation.value` |
| `representation.compression_settings.*` | Ignored for Micro - MAG emits warnings | None |
| `latency_budget.duration.sec/nanosec` | Consumed | `DDS_DataWriterQos::latency_budget.duration` |
| `user_data.value` | Consumed | `DDS_DataWriterQos::user_data.value` |

**Note:**

- Unsupported fields are detected by `ConverterUtils` against `VALID_DATA_WRITER_POLICIES`; MAG warns and ignores unsupported values.
- `unicast` is normalized into `microTransports.enabledTransports` by `SystemModelProducer.addEnabledTransports(...)`; the template emits the resulting transport list.
- `publish_mode.flow_controller_name` is validated by `checkFlowController(...)`; built-in names are accepted directly, and custom names must use the expected prefix and be backed by participant properties.
- `property` is emitted only for names in `VALID_PROPERTIES_TO_GENERATE` (via `QosUtils.microProperties(...)`); other property entries are not emitted into generated C QoS structures.

### 3.18.8 Structural elements mapping

The following mapping documents the structural (non-QoS) elements from the application configuration schema and how MAG processes them.

#### `filter` and `content_filter` (DataReader element)

These define content-based filtering for DataReader subscriptions.

| Schema element | MAG behavior | Notes |
|---|---|---|
| `filter` (deprecated) | Parsed and migrated to `content_filter` with warning | If both `filter` and `content_filter` are present, MAG emits an error |
| `content_filter.kind` | Parsed and validated; only `builtin.sql` is supported | MAG error if kind is not `builtin.sql` or null |
| `content_filter.expression` | Consumed (SQL filter expression string) | Emitted into generated remote-endpoint discovery structs |
| `content_filter.parameter_list` (deprecated) | Parsed and migrated to `expression_parameters` with warning | If both are present, MAG emits an error |
| `content_filter.expression_parameters.element[]` | Consumed (parameter string array) | Emitted alongside filter expression for runtime substitution |

Processing:

- `ConverterUtils.checkContentFilter(...)` handles deprecation migration and validation
- Content filter metadata is emitted in `entities.h` remote data structures via `contentFilterProperty(...)` macro

- Content filter resource limits are configured via `DomainParticipantFactoryQos.filter_qos_policy` properties
- CFT constants: `CFT_DISABLE_WRITER_FILTERING`, `CFT_DISABLE_BUILTIN_SQL_FILTER`, `CFT_FILTER_CLASS_MAX_LENGTH`, `CFT_FILTER_EXPRESSION_MAX_LENGTH`, etc.

#### `register_type` (Domain and Participant element)

Type registration defines how data types are bound to topics.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (required) | Consumed as type identifier | Stored in `RegisterTypeModel.name` |
| `registered_name` | Consumed; defines the type's global name | Stored in `RegisterTypeModel.registeredName` |
| `type_ref` (required) | Resolved against type definitions; used for type plugin lookup | Stored in `RegisterTypeModel.typeRef` |

Processing:

- All `register_type` elements are parsed into `RegisterTypeModel` objects and stored in `DomainParticipantModel.registerTypes`
- Type reference resolution occurs in `SystemModelProducer.findRegisterTypeModel(...)` by matching `name` or `registered_name` against registered types in the domain
- Templates emit type plugin getters via `typePlugin()` and `typePluginCPP()` macros (utils.vm), which select between `registeredName` and `typeRef` for plugin resolution
- If a topic references a type that cannot be resolved, MAG emits an error

#### `multiplicity` (Publisher, Subscriber, DataWriter, DataReader attribute)

The multiplicity attribute specifies how many instances of an entity to generate.

| Schema attribute | MAG behavior | Default value |
|---|---|---|
| `multiplicity` | Consumed as positive integer; used for resource allocation calculations | `1` |

Processing:

- `SystemModelProducer` reads `multiplicity` from Publisher (line 2715), Subscriber (line 2905), DataWriter (line 2805), and DataReader (line 3003)
- `ResourceLimitsUpdater` uses multiplicity to calculate cumulative resource allocations:
  - `Publisher.multiplicity` * `DataWriter` count → total writer allocation

  - `Subscriber.multiplicity` * `DataReader` count → total reader allocation

- Each multiplied instance receives a unique name suffix in generated code

#### base_name and base_names (QoS profile inheritance)

These attributes enable profile inheritance and composition chains.

| Schema element | MAG behavior | Notes |
|---|---|---|
| `base_name` | Parsed as QoS profile reference (single inheritance) | Format: `LibraryName::ProfileName` |
| `base_names.element[]` | Parsed as QoS profile reference chain (composition inheritance) | Multiple profiles resolved in order |

Processing:

- Both are handled by `QosUtils.getBaseName(...)` and `QosUtils.getBaseNames(...)` during QoS resolution
- Format validation: base names must follow `LibraryName::ProfileName` format (checked in `QosUtils.getBaseNamesList(...)`)
- `QosUtils.resolveNonProfileQos(...)` performs recursive inheritance: merges base QoS values with current QoS, with current values overriding inherited ones
- Library prefix is automatically added by `QosUtils.addLibraryNameToBaseName(...)` for relative profile references
- Invalid or unresolved base names generate errors during QoS resolution

#### topic (Domain element)

Topic definitions bind a data type to a topic name with optional QoS.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as topic identifier within domain | Topic name used in `datawriter_qos`/`datareader_qos` topic references |
| `registered_name` (child element) | Alternative topic name if `name` attribute not set | Stored as topic's display/registered name |
| `register_type_ref` (attribute, required) | Resolved against registered types to bind topic to data type | Type plugin lookup |
| `topic_qos` (child element) | Parsed as optional QoS profile for topic-level defaults | Base QoS for DataWriter/DataReader inheritance |

Processing:

- All topics are parsed into `TopicModel` objects and stored per domain
- `SystemModelProducer.findTopicModel(...)` resolves topic references by name (matching against `name` or `registered_name`)
- Topics are primarily used for QoS composition: when a `datawriter_qos` or `datareader_qos` references a topic, inherited fields are merged via `QosUtils.inheritFromTopicQos(...)`
- If a DataWriter/DataReader references a topic that cannot be resolved, MAG emits an error
- Topic QoS is not directly emitted as a C runtime structure; only the resulting merged QoS (after inheritance) is emitted

#### data_writer (Publisher, Participant element)

DataWriter entity configuration.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as entity identifier | Unique writer name within publisher |
| `multiplicity` (attribute) | Resource allocation multiplier (see multiplicity section above) | Instance count for writer generation |
| `topic_ref` (attribute, required) | Resolved against topic definitions in domain | Topic bound to this writer |
| `datawriter_qos` (child element) | Parsed as writer-specific QoS profile | Merged with topic QoS and emitted as `DDS_DataWriterQos` |

Processing:

- `SystemModelProducer.createDataWriterModel(...)` resolves topic references and applies QoS inheritance
- Transport configuration (unicast/multicast) and publish-mode flow-controller validation occur during creation
- Multiplicity is used by `ResourceLimitsUpdater` to calculate writer allocation counts

#### data_reader (Subscriber, Participant element)

DataReader entity configuration, including optional content filtering.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as entity identifier | Unique reader name within subscriber |
| `multiplicity` (attribute) | Resource allocation multiplier (see multiplicity section above) | Instance count for reader generation |
| `topic_ref` (attribute, required) | Resolved against topic definitions in domain | Topic subscribed by this reader |
| `content_filter` or `filter` (child element) | Parsed for content-filtering configuration (see filter section above) | CFT metadata in remote endpoint data |
| `datareader_qos` (child element) | Parsed as reader-specific QoS profile | Merged with topic QoS and emitted as `DDS_DataReaderQos` |

Processing:

- `SystemModelProducer.createDataReaderModel(...)` resolves topic references, applies QoS inheritance, and validates content filters
- `ConverterUtils.checkContentFilter(...)` is called to validate and migrate filter elements
- Multiplicity is used by `ResourceLimitsUpdater` to calculate reader allocation counts
- Content-filter configuration gates the DomainParticipantFactory's CFT feature flag: if any reader has a content filter, `isContentFilter()` returns true

#### publisher (Participant element)

Publisher entity container for DataWriters.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as entity identifier | Unique publisher name within participant |
| `multiplicity` (attribute) | Resource allocation multiplier (see multiplicity section above) | Instance count for publisher generation |
| `publisher_qos` (child element) | Parsed as publisher-specific QoS profile | Emitted as `DDS_PublisherQos` |
| `data_writer` (child elements) | Each DataWriter is processed independently | Collection of writers created for this publisher |

Processing:

- `SystemModelProducer.createPublisherModel(...)` creates a `PublisherModel` and processes child DataWriters
- Publisher multiplicity scales the total number of publishers generated; each multiplied instance receives a unique suffix

#### subscriber (Participant element)

Subscriber entity container for DataReaders.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as entity identifier | Unique subscriber name within participant |
| `multiplicity` (attribute) | Resource allocation multiplier (see multiplicity section above) | Instance count for subscriber generation |
| `subscriber_qos` (child element) | Parsed as subscriber-specific QoS profile | Emitted as `DDS_SubscriberQos` |
| `data_reader` (child elements) | Each DataReader is processed independently | Collection of readers created for this subscriber |

Processing:

- `SystemModelProducer.createSubscriberModel(...)` creates a `SubscriberModel` and processes child DataReaders
- Subscriber multiplicity scales the total number of subscribers generated; each multiplied instance receives a unique suffix

#### domain_participant (ParticipantLibrary element)

Top-level DomainParticipant configuration.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as entity identifier | Participant name prefix for generated code |
| `domain_ref` (attribute, optional) | Resolved against domain library | Binds participant to a specific domain |
| `domain_id` (attribute, optional) | Domain ID override (overrides domain_ref domain ID) | Runtime participant domain ID |
| `base_name` (attribute, optional) | Participant profile inheritance (not directly used for structure) | QoS inheritance base |
| `domain_participant_qos` (child element) | Parsed as participant-specific QoS profile | Emitted as `DDS_DomainParticipantQos` |
| `register_type` (child elements) | Type registrations local to this participant | Stored in participant model; supersede domain registrations |
| `topic` (child elements) | Topics local to this participant | Stored in participant model; supersede domain topics |
| `publisher` / `subscriber` (child elements) | Publishers and subscribers for this participant | Processed recursively |
| `data_writer` / `data_reader` (child elements) | Implicit publisher/subscriber writers and readers | Processed with implicit entity creation |

Processing:

- `SystemModelProducer.createParticipantModel(...)` recursively processes all child entities
- Resource limits are calculated and adjusted by `ResourceLimitsUpdater` based on entity multiplicities and nesting
- Discovery configuration (DPSE or built-in) determines which metadata types are emitted in generated code

#### domain (DomainLibrary element)

Domain definition containing topic and type registrations.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Consumed as domain identifier within library | Domain reference for participants |
| `domain_id` (attribute, optional) | Domain ID for this domain | Runtime domain ID if not overridden |
| `base_name` (attribute, optional) | Domain profile inheritance | QoS inheritance base |
| `register_type` (child elements) | Type registrations available to all participants in domain | Global type registry |
| `topic` (child elements) | Topics available to all participants in domain | Global topic registry |

Processing:

- Domains provide a namespace for types and topics
- Participant domain references resolve to this domain during initialization
- All register types and topics in a domain are merged into participant-local registries; participant-local definitions override domain definitions

#### domain_library and participant_library (Root elements)

Top-level containers for domains and participants.

| Schema element | MAG behavior | Micro target |
|---|---|---|
| `name` (attribute) | Library namespace; used in base_name resolution | Automatically prepended to profile references |
| `domain` (child elements, in domain_library) | Domain definitions | Processed in order |
| `domain_participant` (child elements, in participant_library) | Participant configurations | Processed in order; code generation targets all participants |

Processing:

- Multiple domain and participant libraries can be defined
- Library names are prepended to base_name references by `QosUtils.addLibraryNameToBaseName(...)`
- All participants across all participant libraries are code-generated unless filtering is applied

## 3.19 Building Against FACE Conformance Libraries

This section describes how to build *Connext Micro* using the FACE™ conformance test tools.

### 3.19.1 Requirements

The following items are required for building *Connext Micro* against FACE conformance libraries.

#### Connext Micro source code

RTI distributes the *Connext Micro* source code. Please contact your sales representative(s) for more information.

#### FACE Conformance Tools

RTI does not distribute the FACE conformance tools. Visit https://www.opengroup.org/ for more information.

#### CMake

RTI distributes the *Connext Micro* source with a `CMakeList.txt` project file. CMake is an easy-to-use tool that generates makefiles or project files for various build-tools, such has Linux/macOS makefiles, Microsoft® Visual Studio® project files, and Xcode.

You can download CMake from https://www.cmake.org.

### 3.19.2 FACE golden libraries

The FACE conformance tools use a set of golden libraries. There are different golden libraries for different FACE services, languages and profiles. *Connext Micro only* conforms to the `safetyExt` and `safety` profile of the Operating System Segment (OSS) using the C language.

#### Building the FACE golden libraries

The FACE conformance tools ship with their own set of tools to build the golden libraries. Please follow the instructions provided by FACE. In order to build the FACE golden libraries, you must port them to the required platform.

### 3.19.3 Building instructions

The following instructions describe how to build the *Connext Micro* source against FACE libraries:

- Extract the source code. Please note that the remaining instructions assume that only a single platform is built from the source.
- In the top-level source directory, enter the following command:

```
cmake-gui .
```

  This will start the CMake GUI where all build configuration takes place.

- Click the **Configure** button.
- Select **Unix Makefiles** from the drop-down list.
- Select **Use default compilers** or **Specify native compilers** as required. Then select **Done**.
- Click the **Configure** button again. You should not see any red lines in the output. If you do, click **Configure** again.

  ---

  **Note:** A red line means that a variable has not been configured. Some options could add new variables. Thus, if you change an option, new red lines may appear. In this case configure the variable and press **Configure**.

  ---

- Expand the **CMAKE** and **RTIMICRO** options and configure how to build *Connext Micro*:

| Option | Configuration |
|---|---|
| CMAKE_BUILD_TYPE | Debug or blank.<br>If Debug is used, the *Connext Micro* debug libraries are built. |
| RTIMICRO_BUILD_API | C or C++.<br>C: Include the C API. **For FACE, only C is supported**.<br>C++: Include the C++ API. |
| RTIMICRO_BUILD_API | C or C++.<br>C: Include the C API. **For FACE, only C is supported**.<br>C++: Include the C++ API. |
| RTIMICRO_BUILD_DISCOVERY_MODULE | Dynamic, Static, or Both.<br>Dynamic: Include the dynamic discovery module.<br>Static: Include the static discovery module.<br>Both: Include both discovery modules. |
| RTIMICRO_BUILD_LIBRARY_BUILD | Single or RTI style.<br>Single: Build a single library.<br>RTI style: Build the same libraries RTI normally ships. This is useful if RTI libraries are already being used and you want to use the libraries built from source. |
| RTIMICRO_BUILD_LIBRARY_TYPE | Static or Shared.<br>Static: Build static libraries.<br>Shared: Build shared libraries. |
| RTIMICRO_BUILD_LIBRARY_PLATFORM_MODULE | POSIX |
| RTIMICRO_BUILD_LIBRARY_TARGET_NAME | <Target name><br>Enter a string as the name of the target. This is also used as the name of the directory where the built libraries are placed. If you are building libraries to replace the libraries shipped by RTI, you can use the RTI target name here. It is then possible to set RTIMEHOME to the source tree (if RTI style is selected for RTIMICRO_BUILD_LIBRARY_BUILD). |
| RTIMICRO_BUILD_ENABLE_FACE_COMPLIANCE | Select level of FACE compliance:<br>None: No compliance required.<br>General: Build for compliance with the FACE general profile.<br>Safety Extended: Build for compliance with the FACE safety extended profile.<br>Safety: Build for compliance with the FACE safety profile. |
| RTIMICRO_BUILD_LINK_FACE_GOLDEBLIBS | Check if linking against the static FACE conformance test libraries.<br>**NOTE:** This check-box is only available if FACE compliance is different from None. |
| RTIMICRO_BUILD_LINK_FACE_GOLDEBLIBS | If the RTIMICRO_BUILD_LINK_FACE_GOLDEBLIBS is checked, the path to the top-level FACE root must be specified here. |

- Click **Configure**.
- Click **Generate**.
- Build the generated project.
- Libraries are placed in `lib/<RTIMICRO_BUILD_LIBRARY_TARGET_NAME>`.

## 3.20 Working With Sequences

### 3.20.1 Introduction

*RTI Connext Micro* uses IDL as the language to define data-types. One of the constructs in IDL is the *sequence*: a variable-length vector where each element is of the same type. This section describes how to work with sequences; in particular, the string sequence since it has special properties.

---

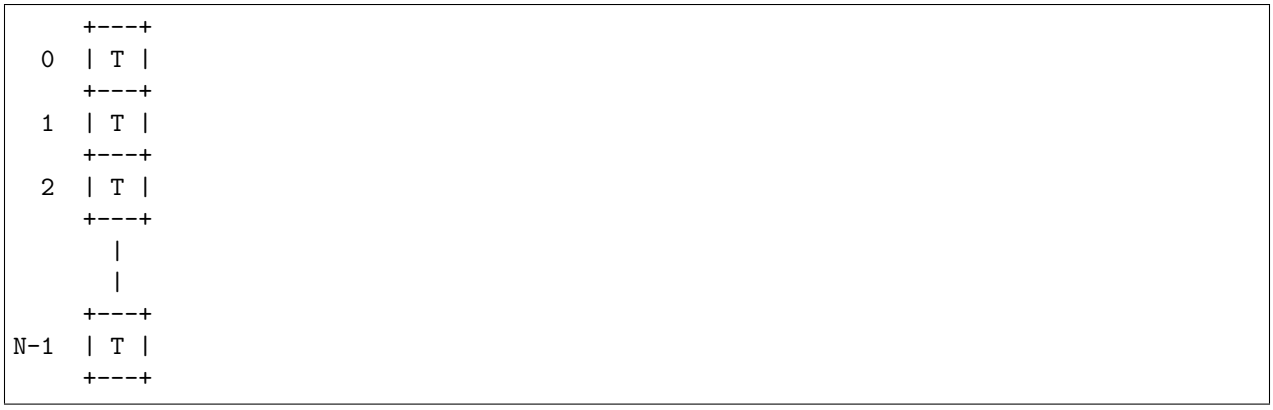
**Note:** This section references several sequence APIs supported by *Connext Micro*. However, *Connext Cert* only supports a subsection of these APIs. Please refer to Sequence Support in the C API Reference for a full list; the APIs supported by *Connext Cert* will have a **«cert»** annotation in their description.

---

### 3.20.2 Working with Sequences

#### Overview

Logically a sequence can be viewed as a variable-length vector with N elements, as illustrated below. Note that sequences indices are 0 based.




There are three types of sequences in *Connext Micro*:

- Builtin sequences of primitive IDL types.
- Sequences defined in IDL using the sequence keyword.
- Sequences defined by the application.

The following builtin sequences exist (please refer to C API Reference and C++ API Reference for the complete API).

| IDL Type | Connext Micro Type | Connext Micro Sequence |
|---|---|---|
| octet | DDS_Octet | DDS_OctetSeq |
| char | DDS_Char | DDS_CharSeq |
| boolean | DDS_Boolean | DDS_BooleanSeq |
| short | DDS_Short | DDS_ShortSeq |
| unsigned short | DDS_UnsignedShort | DDS_UnsignedShortSeq |
| long | DDS_Long | DDS_LongSeq |
| unsigned long | DDS_UnsignedLong | DDS_UnsignedLongSeq |
| enum | DDS_Enum | DDS_EnumSeq |
| wchar | DDS_Wchar | DDS_WcharSeq |
| long long | DDS_LongLong | DDS_LongLongSeq |
| unsigned long long | DDS_UnsignedLongLong | DDS_UnsignedLongLongSeq |
| float | DDS_Float | DDS_FloatSeq |
| double | DDS_Double | DDS_DoubleSeq |
| long double | DDS_LongDouble | DDS_LongDoubleSeq |
| string | DDS_String | DDS_StringSeq |
| wstring | DDS_Wstring | DDS_WstringSeq |

The following are important properties of sequences to remember:

- All sequences in *Connext Micro must* be finite.
- All sequences defined in IDL are sized based on IDL properties and *must* not be resized. That is, *never* call set_maximum() on a sequence defined in IDL. This is particularly important for string sequences.
- Application defined sequences can be resized using set_maximum() or ensure_length().
- There are two ways to use a **DDS_StringSeq** (they are type-compatible):
  - A **DDS_StringSeq** originating from IDL. This sequence is sized based on maximum sequence length *and* maximum string length.
  - A **DDS_StringSeq** originating from an application. In this case the sequence element memory is unmanaged.
- All sequences have an initial length of 0.

### Working with IDL sequences

Sequences that originate from IDL are created when the IDL type they belong to is created. IDL sequences are always initialized with the maximum size specified in the IDL file. The maximum size of a type, and hence the sequence size, is used to calculate memory needs for serialization and deserialization buffers. Thus, changing the size of an IDL sequence can lead to hard to find memory corruption.

The string and wstring sequences are special in that not only is the maximum sequence size allocated, but because strings are also always of a finite maximum length, the maximum space needed for each string element is also allocated. This ensure that *Connext Micro* can prevent memory overruns and validate input.

Some typical scenarios with a long sequence and a string sequence defined in IDL is shown below:

```
/* In IDL */
struct SomeIdlType
{
    // A sequence of 20 longs
    sequence<long,20> long_seq;

    // A sequence of 10 strings, each string has a maximum length of 255 bytes
    // (excluding NUL)
    sequence<string<255>,10> string_seq;
}

/* In C source */
SomeIdlType *my_sample = SomeIdlTypeTypeSupport_create_data();

DDS_LongSeq_set_length(&my_sample->long_seq,5);
DDS_StringSeq_set_length(&my_sample->string_seq,5);

/* Assign the first 5 longs in long_seq */
for (i = 0; i < 5; ++i)
{
    *DDS_LongSeq_get_reference(&my_sample->long_seq,i) = i;
    snprintf(*DDS_StringSeq_get_reference(&my_sample->string_seq,i), 255, "SomeString %d
↪", i);
}

SomeIdlTypeTypeSupport_delete_data(my_sample);

/* In C++ source */
SomeIdlType *my_sample = SomeIdlTypeTypeSupport::create_data();

/* Assign the first 5 longs in long_seq */

my_sample->long_seq.length(5);
my_sample->string_seq.length(5);

for (i = 0; i < 5; ++i)
{
```

(continues on next page)

(continued from previous page)

```
    /* use method */
    *DDS_LongSeq_get_reference(&my_sample->long_seq,i) = i;
    snprintf(*DDS_StringSeq_get_reference(&my_sample->string_seq,i), 255, "SomeString %d
↪", i);

    /* or assignment */
    my_sample->long_seq[i] = i;
    snprintf(my_sample->string_seq[i],255,"SomeString %d",i);
}

SomeIdlTypeTypeSupport::delete_data(my_sample);
```

Note that in the example above the sequence length is set. The maximum size for each sequence is set when my_sample is allocated.

A special case is to copy a string sequence from a sample to a string sequence defined outside of the sample. This is possible, but care *must* be taken to ensure that the memory is allocated properly:

Consider the IDL type from the previous example. A string sequence of equal size can be allocated as follows:

```
struct DDS_StringSeq app_seq = DDS_SEQUENCE_INITIALIZER;

/* This ensures that memory for the strings are allocated upfront */
DDS_StringSeq_set_maximum_w_max(&app_seq,10,255);

DDS_StringSeq_copy(&app_seq,&my_sample->string_seq);
```

If instead the following code was used, memory for the string in **app_seq** would be allocated as needed.

```
struct DDS_StringSeq app_seq = DDS_SEQUENCE_INITIALIZER;

/* This ensures that memory for the strings are allocated upfront */
DDS_StringSeq_set_maximum(&app_seq,10);

DDS_StringSeq_copy(&app_seq,&my_sample->string_seq);
```

#### Working with application defined sequences

Application defined sequences work in the same way as sequences defined in IDL with two exceptions:

- The maximum size is 0 by default. It is necessary to call **set_maximum** or **ensure_length** to allocate space.
- **DDS_StringSet_set_maximum** does not allocate space for the string pointers. The memory must be allocated on a per needed basis and calls to **_copy** may reallocate memory as needed. Use **DDS_StringSeq_set_maximum_w_max** or **DDS_StringSeq_ensure_length_w_max** to also allocate pointers. In this case **_copy** will *not* reallocate

memory.

**Warning:** Mixing API calls that specify the maximum size (i.e., those ending in **__w__max**) with those that do not is *not allowed*. Doing so may cause memory leaks and/or memory corruption.

```
struct DDS_StringSeq my_seq = DDS_SEQUENCE_INITIALIZER;

/* The ensure_length call is _not_ available in Micro Cert */
DDS_StringSeq_ensure_length(&my_seq,10,20);

for (i = 0; i < 10; i++)
{
    *DDS_StringSeq_get_reference(&my_seq,i) = DDS_String_dup("test");
}

/* The finalize call is _not_ available in Micro Cert */
DDS_StringSeq_finalize(&my_seq);
```

**DDS__StringSeq__finalize** automatically frees memory pointed to by each element using **DDS__String__free**. All memory allocated to a string element should be allocated using a **DDS__String** function.

It is possible to assign any memory to a string sequence element if all elements are released manually first:

```
struct DDS_StringSeq my_seq = DDS_SEQUENCE_INITIALIZER;

DDS_StringSeq_ensure_length(&my_seq,10,20);

for (i = 0; i < 10; i++)
{
    *DDS_StringSeq_get_reference(&my_seq,i) = static_string[i];
}

/* Work with the sequence */

for (i = 0; i < 10; i++)
{
    *DDS_StringSeq_get_reference(&my_seq,i) = NULL;
}

DDS_StringSeq_finalize(&my_seq);
```

## 3.21 Example Applications

The RTI *Code Generator* (*rtiddsgen*) can generate example applications from an IDL file, as shown in *Hello World with Publish/Subscribe*. The last type in the IDL is used as the data-type to publish. Refer to *Generating examples* for instructions on how to build these examples.

*Connext Micro* also provides some buildable applications in the installation directory, as described in *Provided examples*.

### 3.21.1 Generating examples

**Note:** Before running *rtiddsgen*, you might need to add `rti_connext_dds-<version>/rtiddsgen/scripts` to your path environment variable folder.

#### Default example

To generate an example, run the following command:

```
rtiddsgen -example -language <C|C++> [-namespace] <file with type definition>
```

This generates an example using the default example template, which uses the Dynamic Participant Dynamic Endpoint (DPDE) discovery plugin.

*rtiddsgen* accepts the following options:

- `-example`: Generates type files, example files, and CMakelists files.
- `-language <C|C++>`: Generates C or C++ code.
- `-namespace`: Enables C++ namespaces when the language option is C++.

The generated example can then be compiled using CMake and the `CMakelists.txt` file generated by *Code Generator*. *Code Generator* also creates a `README.txt` file with a description of the example and instructions for how to compile and run it.

#### Custom example

*Code Generator* can also generate examples using custom templates with the option `-exampleTemplate <templateName>`.

To generate an example using a custom template instead of the default one, run the following command:

```
rtiddsgen -example -exampleTemplate <template name> -language <C|C++> [-namespace] <file␣
↪with type definition>
```

To see the list of the available templates for each language, run the following command:

```
rtiddsgen -showTemplates
```

The output will be similar to the following:

```
List of example templates per language:
    - C:
        - cert
        - dpse
        - shared_memory
        - static_udp
        - waitsets
        - crc
        - zcv2
        - mag/shared_memory
        - mag/static_udp
        - mag/dpde
        - mag/dpse
        - mag/cert/dpse
        - psk
        - content_filter
        - custom_filter
        - cert_custom_filter
        - cert_psk
    - C++:
        - dpse
        - waitsets
        - mag/dpde
    - C++ Namespace:
        - dpse
        - waitsets
```

As an example, the following command will generate an example in the C language, using the `waitsets` custom template instead of the default template:

```
rtiddsgen -example -exampleTemplate waitsets -language C <file with type definition>
```

#### Descriptions of generated examples

Each example consists of a publication and subscription pair to send and receive samples of the type specified by the user. When compiled, the example creates two applications: one to send samples (a publisher) and another to receive samples (a subscriber).

- **default example (no template specified)**

  Discovery of endpoints is done with the *Dynamic Discovery Plugin* (DPDE). Only the *UDP* and *INTRA* transports are enabled. The subscriber application creates a *DataReader*, which uses a listener to receive notifications about new samples and matched publishers. These notifications are received in the middleware thread (instead of the application thread).

- **cert**

  An example that only uses APIs that are compatible with *Connext Cert*.

- **dpse**

  Discovery of endpoints is done with the *Static Discovery Plugin* (DPSE). Static-endpoint discovery uses function calls to statically assert information about remote endpoints belonging to remote *DomainParticipants*.

  The following template variable can be passed with `-V` to configure the generated example:

  – `use_autoenable_entities=false`: Disables auto-enabling of DDS entities on creation. An explicit `Application_enable()` call is generated and invoked after all static discovery assertions are made, so that communication begins only once the application is fully configured.

- **shared_memory**

  The only transport used is shared memory. Because the *UDP Transport* is disabled and only the *Shared Memory Transport (SHMEM)* is enabled, both the publisher and subscriber applications need to run in the same operating system instance.

- **static_udp**

  This example uses a static *UDP Transport* interface configuration. Using this API, the UDP transport is statically configured. This is useful in systems that are not able to return the installed UDP interfaces (name, IP address, mask, etc.).

- **waitsets**

  In this example, the Subscriber application creates a *DataReader* that uses a Waitset (instead of a listener) to receive notifications about new samples and matched publishers. These notifications are received in the middleware thread (instead of the application thread).

- **crc**

  This example includes configuration of the *Message Integrity Checking* settings. The CRC fields in the WireProtocolQosPolicy enable the generation of a checksum value based on the data being transmitted. They also determine which checksums are allowed and whether they should be sent.

- **zcv2**

  This example uses the *Zero Copy v2 Transport* and handles discovery over the *UDP Transport*. User traffic is handled using the Zero Copy v2 transport.

- **mag/**

  These examples create an example XML application file and an example Qos profile file and then use *Application Generation Using XML* to generate the application from the XML and Qos file.

  The following template variable can be passed with `-V` to configure the generated example:

  – `use_autoenable_entities=false`: Disables auto-enabling of DDS entities on creation. An explicit `Application_enable()` call is generated and invoked after all static discovery assertions are made, so that communication begins only once the application is fully configured.

- **psk**

  This example uses pre-shared key (PSK) encryption to secure communications. You must have the *Lightweight Security Plugin* installed in order to compile this example.

- **content_filter**

  This example demonstrates how to use content filtering with the builtin SQL content filter and *Dynamic Discovery Plugin* (DPDE). The subscriber application configures the *DataReader* QoS to filter samples using an SQL expression, and the publisher application enables writer-side filtering.

- **custom_filter**

  This example demonstrates how to use content filtering with a custom content filter and *Dynamic Discovery Plugin* (DPDE). The subscriber application configures the *DataReader* QoS to filter samples using a custom content filter, and the publisher application enables writer-side filtering.

- **cert_custom_filter**

  This example demonstrates how to use content filtering with a custom content filter, *Connext Cert*, and *Static Discovery Plugin* (DPSE). The subscriber application configures the *DataReader* QoS to filter samples using a custom content filter. The publisher application enables writer-side filtering and manually asserts the content filter parameters used by the subscriber.

- **cert_psk**

  This example only uses APIs compatible with *Connext Cert* and uses pre-shared key (PSK) encryption to secure communications. Discovery of endpoints is done with the *Static Discovery Plugin* (DPSE). You must have the *Lightweight Security Plugin* installed in order to compile this example.

#### How to compile and run the generated examples

Before compiling, set the environment variable `RTIMEHOME` to the *Connext Micro* installation directory. Your *Connext Micro* directory includes `rtime-make` (on Linux® and macOS® systems) or `rtime-make.bat` (on Windows® systems) to simplify invocation of CMake. This script requires `RTIMEHOME` to be set and is a convenient way to invoke CMake with the correct options.

Consult the `README.txt` file in the generated example directory for detailed instructions on compiling and running your application.

> **Warning:** We recommend using the toolchain file that matches the target architecture to compile the generated examples.
>
> For example, if the target architecture is `--target armv8leElfgcc13.3.0-Linux6`, then the example applications should be compiled with the `armv8leElfgcc13.3.0-Linux6` toolchain file. Failing to do so may cause warnings.

### 3.21.2 Provided examples

*Connext Micro* provides buildable example applications in the `example/` directory of your installation. Each example demonstrates different features, as described below:

- **HelloWorld_transformations**: A HelloWorld example that uses *UDP transformations* to send encrypted packets using OpenSSL.
- **HelloWorld_dgram**: A HelloWorld example that uses an example *DGRAM* implementation.

Consult the `README.txt` file included with each example for instructions on how to build and run the application.

## 3.22 Lightweight Security Plugin

The *Lightweight Security Plugin* is an addon for *Connext Micro* that offers an efficient solution for securing RTPS communication at the domain level. This plugin protects your domain from unauthorized participants and ensures the integrity and confidentiality of RTPS messages.

### 3.22.1 Overview

The *Lightweight Security Plugin* uses symmetric encryption, deriving keys from a combination of publicly-available data and a user-provided pre-shared key (PSK) seed. Pre-shared key protection offers metadata and data protection to RTPS messages and restricts communication only to *DomainParticipants* holding the correct PSK. This architecture offers a streamlined path to deploy security without the overhead of full public key infrastructure.

The plugin also includes a *passphrase tracker*, a component that periodically checks for updates to the PSK passphrase. This feature allows you to rotate the passphrase without restarting your application.

The *Lightweight Security Plugin* consists of a Platform Independent Library (PIL) and a Transform Platform Support Library (PSL). The Transform PSL is an abstraction layer that allows seamless integration with a cryptographic backend. The plugin includes a reference implementation that integrates with OpenSSL 3.5[1] and a file-based provider for the passphrase tracker.

This modular design makes the *Lightweight Security Plugin* an ideal choice for applications that demand robust protection without compromising on portability or efficiency.

[1] You can substitute in your own Transform PSL to meet performance, certification, or compliance requirements. For details, please contact RTI through support.rti.com.

### 3.22.2 Downloading and Installing the Lightweight Security Plugin

Once you have access to the *Lightweight Security Plugin*, you can download the following package files from support.rti.com:

- `rti_connext_dds_micro-<version>-lw-security-host.rtipkg`
- `rti_connext_dds_micro-<version>-lw-security-target-openssl-3.5-<architecture>.rtipkg`
- `openssl-3.5-micro-<version>-target-<architecture>.rtipkg` (*Optional*)

Download the host package with `<version>` 4.3.0, and a target package with `<version>` 4.3.0 and `<architecture>` matching your CPU and compiler.

To install the *Lightweight Security Plugins* packages, follow the same instructions for installing *Connext Micro* packages in *Installing Connext Micro*.

#### Package contents

The host package contains the necessary header files for lightweight security development.

The target package provides the Platform Independent Library (PIL) and Transform Platform Support Library (PSL) specific to your target architecture. The Transform PSL is dependent on OpenSSL; RTI builds and tests the PSL against OpenSSL 3.5. OpenSSL 3.5.

The OpenSSL target package simply contains the OpenSSL 3.5 release. RTI provides this for convenience, but you may use any OpenSSL distribution ABI compatible with 3.5.4. The OpenSSL source can be found at https://openssl-library.org/source/.

The package also includes a reference implementation of the passphrase tracker provider that reads the passphrase from a file.

### 3.22.3 Getting Started

This section will walk you through the steps needed to enable the *Lightweight Security Plugin* in your application and run it in your domain. We will use the provided reference implementation of the Transform PSL with OpenSSL 3.5 and the file-based provider for the passphrase tracker.

---

**Note:** You can also generate an example application with *rtiddsgen* that uses lightweight security. Refer to *Example Applications* for more information.

---

### Include header files

The following header files are required to use the *Lightweight Security Plugin*:

- `dds_psk/dds_psk.h`: provides the functions and types needed to register the *Lightweight Security Plugin* with *Connext Micro*.
- `rti_me_psk/pskpsl/psk_ossl_transform.h`: provides the function `PSK_OSSL_get_interface` to retrieve the reference Transform PSL interface implementation based on OpenSSL.
- `rti_me_psl/pskpsl/psk_file_pass_tracker.h`: provides the function `PSK_FilePassTracker_get_interface` to retrieve the reference *file passphrase tracker* provider interface.

### Link application and libraries

To link an application with the lightweight security feature, the following libraries are required in the listed order:

- `RTIMEHOME/lib/<arch>/`
  1. `rti_me_ddspsk` (must be linked before `rti_me`)
- `RTIMEHOME/lib/<arch>-<PSL>/`
  2. `rti_me_pskpsl_openssl3` (must be linked before `rti_me`)
- `RTIMEHOME/thirdparty/openssl-3.5.4-<arch>/<debug|release>-<shared|static>/lib`
  3. `libcrypto`

### Enable lightweight security in your application

To use lightweight security, you must first register the *Lightweight Security Plugin* library with *Connext Micro* via the DDS_PskLibrary_register() function, as shown below:

```
{
     struct DDS_PskServiceFactoryProperty psk_svc_property =
         DDS_PskServiceFactoryProperty_INITIALIZER;
     psk_svc_property.psl_get_interface_func = PSK_OSSL_get_interface;
     psk_svc_property.pass_tracker_get_interface_func = PSK_FilePassTracker_get_
↪interface;
     if(!DDS_PskLibrary_register(
                           registry,
                           &psk_svc_property))
     {
       printf("Lightweight Security Plugin registration error\n");
     }
}
```

**Note:** When using the C++ API, register the *Lightweight Security Plugin* using the same method as the C API, shown above.

This function takes a property structure with the following fields:

- service_name: a unique identifier for this registration within the *DomainParticipantFactory*. The *DomainParticipant* QoS references this name to select which library registration to use. If set to NULL, the default name `DDS_PSK_DEFAULT_SUITE_NAME` will be used.
- psl_get_interface_func: a function pointer used to retrieve the Transform PSL interface. When using the reference Transform PSL provided by RTI, set this to `PSK_OSSL_get_interface`[2].
- psl_config: An opaque pointer passed to the Transform PSL creation function. When using the RTI-provided Transform PSL, this field resolves to DDS_SecTransform_Configuration. Its type is `struct DDS_SecTransform_Configuration *`, as shown in the example below:

```
struct DDS_SecTransform_Configuration config = *PSK_OSSL_get_default_configuration();
```

- pass_tracker_get_interface_func: a function pointer used to retrieve the passphrase tracker provider interface. When using the reference *file passtracker* provided by RTI, set this to `PSK_FilePassTracker_get_interface`[3].
- pass_tracker_config: An opaque pointer passed to the passphrase tracker creation function. When using the RTI-provided *file passtracker*, this field resolves to PSK_PassTrackerConfiguration. Its type is `struct PSK_PassTrackerConfiguration *`, as shown in the example below:

```
struct PSK_PassTrackerConfiguration pass_tracker_config =
    PSK_FilePassTracker_get_default_configuration();
```

### Specify the plugin in the DomainParticipant QoS

Assign the service_name that you specified during registration to the `DomainParticipantQos.trust.suite` field with the following function:

```
RT_ComponentFactoryId_set_name(&dp_qos.trust.suite, DDS_PSK_DEFAULT_SUITE_NAME)
```

This function tells the *DomainParticipant* to enable the *Lightweight Security Plugin* and identifies which registered instance to use.

[2] This function is provided by `rti_me_psl/pskpsl/psk_ossl_transform.h`.
[3] This function is provided by `rti_me_psl/pskpsl/psk_file_pass_tracker.h`.

### Configure the plugin

The *Lightweight Security Plugin* accepts the following properties (in the PROPERTY QoS):

- `dds.sec.crypto.rtps_psk_secret_passphrase` (*Required*)
- `dds.sec.access.rtps_psk_protection_kind` (*Optional*)
- `dds.sec.crypto.rtps_psk_symmetric_cipher_algorithm` (*Optional*)
- `com.rti.serv.secure.cryptography.max_blocks_per_session` (*Optional*)
- `com.rti.serv.secure.pass_tracker.period_sec` (*Optional*)
- `com.rti.serv.secure.pass_tracker.period_nsec` (*Optional*)
- `com.rti.serv.secure.pass_tracker.provider_arg` (*Optional*)

For more information about these properties, refer to *Configuring the Lightweight Security Plugin*. For now, set these properties as shown in the example snippet below:

```
{
  struct DDS_DomainParticipantQos dp_qos =
  DDS_DomainParticipantQos_INITIALIZER;

  /* Required property*/
  DDS_PropertyQosPolicyHelper_add_property(
      &dp_qos.property,
      "dds.sec.crypto.rtps_psk_secret_passphrase",
      "data:,404166165:castle super radar denial swing lunar kind swarm wet toilet␣
↪output harbor basic begin margin huge year visit",
      DDS_BOOLEAN_FALSE);

  /* Optional Properties */
  DDS_PropertyQosPolicyHelper_add_property(
      &dp_qos.property,
      "dds.sec.access.rtps_psk_protection_kind",
      "ENCRYPT",
      DDS_BOOLEAN_FALSE);

  DDS_PropertyQosPolicyHelper_add_property(
      &dp_qos.property,
      "dds.sec.crypto.rtps_psk_symmetric_cipher_algorithm",
      "AUTO",
      DDS_BOOLEAN_FALSE);

  DDS_PropertyQosPolicyHelper_add_property(
      &dp_qos.property,
      "com.rti.serv.secure.cryptography.max_blocks_per_session",
      "256",
      DDS_BOOLEAN_FALSE);

  /* PassTracker properties */
  DDS_PropertyQosPolicyHelper_add_property(
      &dp_qos.property,
```

(continues on next page)

(continued from previous page)

```
      "com.rti.serv.secure.pass_tracker.provider_arg",
      "file:./passphrase.txt",
      DDS_BOOLEAN_FALSE);

  DDS_PropertyQosPolicyHelper_add_property(
      &dp_qos.property,
      "com.rti.serv.secure.pass_tracker.period_sec",
      "5",
      DDS_BOOLEAN_FALSE);
}
```

#### Create a protected DomainParticipant

Invoke the function DDS_DomainParticipantFactory_create_participant with the QoS `dp_qos`.

At this point, your application is protected by the *Lightweight Security Plugin* and ready to run!

### 3.22.4 Configuring the Lightweight Security Plugin

The following table lists the properties used to configure the *Lightweight Security Plugin* and detailed descriptions of their use. Refer to the OMD DDS Security Specification 1.1 for additional information.

All of the properties below have string values.

| Property Name | Property Description |
|---|---|
| `dds.sec.crypto.rtps_psk_secret_passphrase` | Passphrase used to derive the per-participant key used for encoding RTPS messages (in combination with other publicly available data).<br>This property is mutable through the *DomainParticipant* `set_qos` API. The *DomainParticipant* will use the updated passphrase to derive the new local and remote pre-shared keys. From that point on, the *DomainParticipant* will use the new keys to encrypt and decrypt RTPS messages with PSK protection. You should expect warning messages (about receiving messages encoded with a different pre-shared key identifier) as you change the property value throughout your system. Once all your *DomainParticipants* have the same passphrase, all PSK protected traffic will be encoded and decoded with consistent keys. At that point, warning messages will stop.<br>Updating the value of this property implies a change in the secret key AND in the key identifier. Each secret key should be associated with a unique key identifier. Updating the secret key without changing the key identifier is not allowed and will result in set_qos returning `DDS_RETCODE_NOT_ALLOWED_BY_SECURITY`.<br>The pre-shared secret identifier should not be reused; you may set a new secret key with an old key identifier, but doing so is not advised. In that case, *Lightweight Security Plugin* may end up using the old pre-shared key to try (unsuccessfully) to decode an RTPS message that was encoded using the new key.<br>The passphrase must follow the format `data:,<passphrase_id>:<passphrase>`.<br>`<passphrase_id>` is a unique four-byte integer number that identifies the key. It must be in the [0, 2^32 - 1] range, excluding the values that have 0xFF as the least significant byte (e.g., 255, 1023, 2047, etc.). `<passphrase_id>` must be provided to support proper handling of different passphrase revisions, and it must be consistent across all the *DomainParticipants* within a domain. This value must be different with every passphrase change, otherwise the update will be rejected. RTI recommends using incremental subsequent values for every new key provided. Rolling back to a lower number (e.g., 0) after the indexes have been exhausted is acceptable.<br>`<passphrase>` is the secret passphrase string that the *Lightweight Security Plugin* will use to derive the per-participant pre-shared key. `<passphrase>` must contain up to 512 ASCII printable characters (character codes 32 to 126, both included). The first and last characters of the `<passphrase>` shall not be the space character (character code 32).<br>**Default**: not set<br><br>**Attention:** It is your responsibility to provide a key seed with good entropy and length. Ideally, the pre-shared key seed should be a true random 256-bit seed. There are a |

### 3.22.5 Technical Considerations

Consider the following when using the *Lightweight Security Plugin*:

#### RTPS transport only

The *Lightweight Security Plugin* only protects transports that use the RTPS protocol. That means that INTRA and Zero Copy V2 transports cannot be protected with the *Lightweight Security Plugin*.

#### Maximum Transmission Unit (MTU)

RTI limits the maximum size of a single packet to 65535 bytes. If using a transport that can support an MTU greater than this, the size will be truncated to 65535 bytes.

---

**Note:** Types that exceed this limit can still be fragmented and sent by *Connext Micro*. This only limits the size of individual packets.

---

### 3.22.6 Interoperability

The *Lightweight Security Plugin* interoperates with the *Connext Professional Lightweight Builtin Security Plugins* security solution out of the box, with some considerations explained in the following sections.

The *Lightweight Security Plugin* is also compatible with *Connext Professional*'s *Builtin Security Plugins* when the *Builtin Security Plugins* are configured for compatibility; refer to Lightweight Builtin Security Plugins and Builtin Security Plugins Interoperability for more information.

#### Additional Authenticated Data (AAD)

The *Lightweight Security Plugin* always uses AAD, which adds a header extension to RTPS messages. *Connext Professional* allows you to disable this. For successful interoperability between *Connext Micro* and *Connext Professional*, you must configure *Connext Professional* to always enable AAD by setting the cryptography.enable_additional_authenticated_data property to `TRUE`.

#### Passphrase ID limitations

*Connext Micro* supports a `passphrase_id` (which is part of the passphrase) up to `MAX_UINT64`. However, *Connext* 7.3.0 and above only supports a `passphrase_id` up to 254. To ensure interoperability between *Connext Micro* and *Connext Professional*, choose a `passphrase_id` within the range of 0 to 254.

## 3.23 Memory Management

*Connext Micro* is designed for use in real-time systems and uses a predictable, deterministic memory manager to ensure that memory growth is not unbounded, OS memory fragmentation is eliminated, and memory usage can be determined before runtime. This design ensures proper operation after all entities have been created, but requires the system designer to properly configure *Connext Micro*, as described in this section.

### 3.23.1 Resource limits

*Connext Micro* uses resource limits to determine how much to allocate of a particular resource; e.g., the maximum number of samples that can be cached by a *DataReader*, or how many *DataWriters* can be created locally. These limits set the maximum amount of memory that can be dynamically allocated, and may incur additional overhead.

This section provides guidance on how much overhead these resource limits incur. Simple resource limits, such as those that specify a number of bytes, are not discussed here.

Please refer to Resource Limits for the full list of resource limits, and to RTI Connext Performance Benchmarks on RTI Community for memory usage benchmarks.

### 3.23.2 Dynamic memory allocation

*Connext Micro* allocates heap memory to create internal data structures dynamically as DDS entities are created. Keep in mind that *Connext Micro* manages allocated memory using its own internal memory management, and *only* returns freed allocated memory when something is deleted. So, if an application never deletes anything, no memory is ever freed.

---

**Note:** *Connext Micro* uses the term "heap" to refer to any memory that is allocated dynamically, typically using the `malloc` system call. However, the Platform Support Library (PSL) is responsible for implementing the memory allocation and freeing functions required by *Connext Micro*; the only requirement is that heap memory is valid from when it is allocated to when it is freed.

---

**Attention:** *Connext Micro* does not support dynamically allocating resources beyond the initial configuration. Therefore, all resource limits must be finite.

### 3.23.3 DDS resource limits

The following table shows which resource limits are applicable to DDS APIs that allocate memory:

Table 3.14: DDS APIs that allocates memory

| API | Creates | Applicable Resource Limits |
|---|---|---|
| DDS_DomainParticipantFactory_get_instance | DDS_DomainParticipantFactory | DDS_DomainParticipantFactoryQos OSAPI_LogProperty OSAPI_SystemProperty |
| DDS_DomainParticipantFactory_create_participant | DDS_DomainParticipant | DDS_DomainParticipantQos |
| DDS_DomainParticipant_create_topic | DDS_Topic | Topic name length |
| DDS_Publisher_create_datawriter | DDS_DataWriter | DDS_DataWriterQos |
| DDS_Subscriber_create_datareader | DDS_DataReader | DDS_DataReaderQos |
| DDS_WaitSet_new | DDS_WaitSet | None |
| FooTypeSupport_register_type | TypeSupport | Type name length *Discovery plugin resource limits* |

### 3.23.4 Discovery plugin resource limits

*Connext Micro* preallocates memory to store discovery information. If these limits are exceeeded, *Connext Micro* discards the discovery information. *Connext Micro* includes two different discovery plugins:

1. Dynamic Participant Dynamic Endpoint (DPDE). Please refer to DPDE for details. DPDE is the simplest discovery plugin to use; however, its lack of ability to ignore DDS entities may cause *Connext Micro* to run out of resources if running in a non-determinstic environment. For this reason, DPDE is recommended when running in a deterministic environment OR when resource limits are set sufficiently high to accomodate all possible scenarios.
2. Dynamic Participant Static Endpoint (DPSE). Please refer to DPSE for details. DPSE trades off ease of use, higher memory usage, and non-deterministic behavior with more upfront configuraton, lower memory usage, and deterministic behavior.

The DDS discovery process does not only involve storing discovery information; it also involves matching local endpoints with discovered endpoints. This means that a single local *DataWriter* must allocate memory to track every *DataReader* discovered in the domain, regardless of whether the remote *DataReader* subscribes to the same *Topic*. If the domain contains hundreds or thousands of unrelated endpoints, the local *DataWriter* must still maintain memory for each one to determine if they are a match. As a result, local resource limits (such as the maximum number of tracked entities) can be exhausted by a single endpoint. This can prevent communication from functioning correctly, even in a simple system.

The following resource limits are important to take into account:

Table 3.15: Resource Limits and Discovery

| Resource Limit | Description |
|---|---|
| `DDS_DomainParticipantQos.resource_limits.remote_participant_allocation` | The maximum number of *DomainParticipants* that can be discovered. The *DomainParticipant* itself is never discovered. |
| `DDS_DomainParticipantQos.resource_limits.remote_reader_allocation` | The maximum number of remote *DataReaders* that can be discovered. Locally created *DataReaders* are not discovered via discovery plugin and should *not* count toward this limit. |
| `DDS_DomainParticipantQos.resource_limits.remote_writer_allocation` | The maximum number of remote *DataWriters* that can be discovered. Locally created *DataWriters* are not discovered via discovery plugin and should *not* count toward this limit. |
| `DDS_DomainParticipantQos.resource_limits.matching_writer_reader_pair_allocation` | The maximum number of RTPS sessions that can be created. An RTPS session is created between a *DataReader* and a matched *DataWriter and* between a *DataWriter* and a matched *DataReader*. |
| `DDS_DataWriterQos.writer_resource_limits.max_remote_readers` | The maximum number of *DataReaders* the *DataWriter* can match with. Note that both discovered *and* locally created *DataReaders* should count toward this resource limit. |
| `DDS_DataReaderQos.writer_resource_limits.max_remote_writers` | The maximum number of *DataWriters* the *DataReader* can match with. Note that both discovered *and* locally created *DataWriters* should count toward this resource limit. |

### 3.23.5 Type support resource limits

*Connext Micro* supports two different types of code generation from IDL:

1. **Interpreted**: The is the default and requires the use of the `rti_me_ddsxtypes` library. However, it also allocates additional memory. You can enable interpreted type support by passing `interpreted 1` to *rtiddsgen*.

2. **Non-Interpreted**: The non-intepreted type support does not support the *DDS XTypes* specification. However, it uses significantly less memory. This is also the only type support supported by the CERT profile. You can enable the non-interpreted type support by passing `interpreted 0` to *rtiddsgen*.

## 3.24 Transport Priority

The TRANSPORT_PRIORITY QoS policy allows you to set a priority for the data being sent out by a *DataWriter* or a *DataReader*. This QoS policy is optional and only supported on certain operating systems and transports.

Table 3.16: DDS_TransportPriorityQosPolicy

| Type | Field Name | Description |
|---|---|---|
| `DDS_Long` | `value` | Hint as to how to set the data priority. |

*Connext Micro* will propagate the `value` set on a per-*DataWriter* or per-*DataReader* basis to the transport when the *DataWriter* publishes data. It is up to the implementation of the transport to do something with the `value`, if anything.

The DDS specification does not define how implementations must handle data with different priorities; in many cases, a DDS implementation cannot treat higher-priority data differently from lower-priority data. This limitation is especially common when the same thread that calls the *DataWriter*'s `write()` operation also sends the data directly to the physical transport. Also, many physical network transports do not provide user-level control over packet priority.

In *Connext Micro*, for the builtin UDP transport, the `value` set in the TRANSPORT_PRIORITY QoS policy can be used to set the differentiated services (DS) field bits of the IPv4 header for datagrams sent by a *DataWriter* or *DataReader*. It is platform-dependent on how and whether setting the DS field has an effect. Some platforms may require external permissions in order to set the DS field.

Do not assume that enabling the TRANSPORT_PRIORITY QoS policy will affect end-to-end data delivery between a *DataWriter* and a *DataReader*. For priority handling to work, all network elements (such as switches and routers) must support and actively use the DS field to differentiate packet priority. Therefore, you must design and configure transport priority at the system level. Enabling it in the application alone may have no effect on transport behavior.

### 3.24.1 Using transport priority with UDP

In *Connext Micro*, the `value` you set in the TRANSPORT_PRIORITY QoS policy can control the DS field in the IPv4 header. The UDP transport does not use the transport priority value directly; instead, it transforms the value according to the following properties in UDPInterfaceFactoryProperty:

- transport_priority_mask
- transport_priority_mapping_low
- transport_priority_mapping_high

Conceptually, the mapping works as follows:

1. `transport_priority_mask` selects which bits of the TRANSPORT_PRIORITY participate in the mapping.

2. The selected value is then linearly scaled into the output range defined by `transport_priority_mapping_low` and `transport_priority_mapping_high`.
3. The resulting value is written into the DS/TOS field of the outgoing IP packet for IP-based transports.

---

**Note:** The mapping performs a linear scaling of the selected priority bits into the configured `[transport_priority_mapping_low, transport_priority_mapping_high]` range. While this mechanism can be used to expand or compress an application's transport priority space, modern networks typically use specific DSCP codepoints to represent traffic classes. In such cases, applications may prefer to configure `transport_priority_mask = 0xff` and directly specify the desired DS value in the TRANSPORT_PRIORITY.

---

The mapping is computed as follows:

- Let n be the number of 4-bit right shifts needed until (mask & 0xffff0000) == 0, then:
  - masked_prio = (transport_priority_in & mask) » (n × 4)
  - mask' = mask » (n × 4)
  - IP_TOS = (masked_prio × range / mask') + transport_priority_mapping_low
- where range = `transport_priority_mapping_high - transport_priority_mapping_low`.

To make the transport priority value map directly to the DS field (i.e., use the priority value as-is), set the transport_priority_mask to `0xFF` and leave the other properties at their default values.

### 3.24.2 Configuring transport priority

When registering the *UDP Transport*, you can set the transport_priority_mask and other values as shown in the following snippet:

```
struct UDP_InterfaceFactoryProperty *udp_property = NULL;

  /* Allocate a property structure for the heap, it must be valid as long
   * as the component is registered
   */
  udp_property = (struct UDP_InterfaceFactoryProperty )
                      malloc(sizeof(struct UDP_InterfaceFactoryProperty));
  if (udp_property != NULL)
  {
      udp_property = UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;
       /* using the transport priority
          as defined in the Transport Priority QoS policy */
       udp_property->transport_priority_mask = 0xFF;
       udp_property->transport_priority_mapping_high = 0xFF;
       udp_property->transport_priority_mapping_low = 0;
       /* create two send sockets in total */
       udp_property->max_unicast_send_sockets = 2;
      /* Register the transport
```

(continues on next page)

(continued from previous page)

```
     */
    if (!RT_Registry_register(registry, "_udp",
              UDP_InterfaceFactory_get_interface(),
              (struct RT_ComponentFactoryProperty*)udp_property,
               NULL))
    {
        /* ERROR */
    }
}
else
{
    /* ERROR */
}
```

Then, for each *DataWriter* and *DataReader*, configure the transport priority using the TRANSPORT_PRIORITY QoS policy with a value between 0 and 0xFF. The priorities on *DataWriters* and *DataReaders* are independent; the priority set on a *DataWriter* will set the DS field for RTPS packets sent by the *DataWriter*, while the priority set on a *DataReader* will set the Type of Service (ToS) field for RTPS packets sent by the *DataReader*.

For example, if you want to prioritize the traffic sent by a *DataWriter* using the Expedited Forwarding (EF) traffic class, your priority should be set to the following:

```
struct DDS_DataWriterQos dw_qos = DDS_DataWriterQos_INITIALIZER;
dw_qos.transport_priority.value = 0XB8;
```

Where `0xB8` = `101110 00`, and `101110` is the Differentiated Services Code Point (DSCP) and `00` is the Explicit Congestion Notification (ECN).

See the Wireshark capture snapshot below showing how the DS field was set:

### 3.24.3 Examples

The following examples illustrate how different configurations affect the resulting priority value provided to the transport.

Table 3.17: Transport Priority Examples

| TRANSPORT_PRIORITY value | mask | mapping_low | mapping_high | Resulting IP_TOS | Notes |
|---|---|---|---|---|---|
| 0xB8 | 0xFF | 0 | 0xFF | 0xB8 | No scaling (direct mapping) |
| 0x00 | 0x03 | 0 | 0x03 | 0x00 | Lowest priority |
| 0x01 | 0x03 | 0 | 0x3F | 0x15 | Priority scaled into range |
| 0x02 | 0x03 | 0 | 0x3F | 0x2A | Higher priority |
| 0x03 | 0x03 | 0 | 0x3F | 0x3F | Highest priority |

### 3.24.4 Enforcing priority selection

By default, *Connext Micro* will enforce the priority assigned to the *DataWriter* or *DataReader*. If the system call to change the priority on the socket fails, the message will fail to send. However, you can change this by compiling the PSL with `RTIME_UDP_ENFORCE_TRANSPORT_PRIORITY` set to false. Refer to *Building the PSL* for compilation instructions.

### 3.24.5 System resource and performance considerations

By default, *Connext Micro* uses a single send socket when you use the reference *UDP Transport*. If multiple *DataWriters* and *DataReaders* use different transport priorities, this configuration can negatively affect throughput.

You can increase the number of unicast send sockets by setting max_unicast_send_sockets to a value greater than 1. *Connext Micro* will create the specified number of sockets and attempt to use them efficiently based on message priority. For the best performance, set max_unicast_send_sockets to a value equal to the total number of distinct transport priorities used by all *DataWriters* and *DataReaders* in the system. This count must include both user-defined entities and builtin *DataWriters* and *DataReaders*.

---

**Note:** Keep in mind that increasing the number of sockets improves priority separation but also increases system resource usage.

---

## 3.25 Startup Time Guidelines

This section provides guidelines to measure and reduce initialization and discovery time in *Connext Micro*. These tips apply to any deployment where startup time matters, such as automotive or industrial systems with strict startup budgets.

### 3.25.1 Understanding the startup process

Startup in *Connext Micro* has four phases:

1. Creation of local DDS entities (*DomainParticipants*, *Topics*, *DataWriters*, and *DataReaders*);
2. Enabling those entities;
3. Discovery processing, where your *DomainParticipants* exchange discovery traffic with remote *DomainParticipants*;
4. Discovery completion, where your application has discovered all expected remote *DomainParticipants* and endpoints.

The autoenable_created_entities setting controls whether these phases overlap. When autoenable is TRUE, entities are enabled as they are created, so discovery can begin before all entities exist; phases 1, 2, and 3 run partially in parallel. When autoenable is FALSE, you create all entities first (disabled), then enable the *DomainParticipant*. Discovery begins only after the *DomainParticipant* is enabled.

This section separates startup into two measurable intervals: initialization time (phases 1 and 2) and discovery time (phases 3 and 4). These intervals can be tuned independently.

#### Initialization time

This is the interval from application start until all required local DDS entities are created and enabled. This includes creating and enabling the *DomainParticipant* and any locally required *Topics*, *DataWriters*, *DataReaders*, and associated local resource allocation.

When autoenable is TRUE, discovery may overlap with initialization Time. It excludes waiting for remote discovery or matching and any platform or network bring-up outside DDS initialization.

#### Discovery time

This is the interval from when the *DomainParticipant* is enabled until your application has discovered the expected set of remote *DomainParticipants* and endpoints. It includes *DomainParticipants* discovery and endpoint discovery traffic and processing, but excludes protocols above DDS discovery unless explicitly stated.

Application-installed listeners (e.g., `on_subscription_matched`, `on_publication_matched`) run on discovery threads and can increase discovery time if they perform expensive operations, such as logging.

### 3.25.2 Minimizing startup time

Slow startup is commonly caused by unoptimized builds, blocking I/O during startup, or oversized resource limits. The following sections describe application design choices, QoS configuration, and build settings that affect startup time.

---

**Note:** Some platforms have specific startup-time guidance. See *Platforms Guide* for details on your target platform.

---

#### Application design

The following subsections describe application-level design decisions that can affect startup time.

#### Participant and domain architecture

Each *DomainParticipant* adds initialization overhead and consumes internal resources. Use one *DomainParticipant* per application as the default approach, and only introduce additional *DomainParticipants* when you need to communicate in multiple DDS domains.

If your system has groups of applications that do not need to communicate with each other, place them in separate DDS domains to reduce discovery traffic and prevent exceeding discovery resources (e.g., the maximum number of remote entities).

For a general overview of *DomainParticipants* and the discovery process, see *Discovery*.

#### Enabling entities manually

When many entities are created at startup, enabling them incrementally or relying on autoenable increases initialization time and may impact discovery time determinism. If your application design permits it, create all DDS entities disabled (set autoenable_created_entities to FALSE) and call DDS_Entity_enable() with the following syntax after all entities are created:

```
DDS_Entity_enable(DDS_DomainParticipant_as_entity(participant))
```

#### Discovery plugin selection

*Connext Micro* provides two discovery plugins with different startup characteristics: *DPDE (Dynamic Participant Dynamic Endpoint)* and *DPSE (Dynamic Participant Static Endpoint)*.

DPSE has a smaller base footprint (~14 KB) than DPDE (~48 KB) because DPDE creates three pairs of built-in *DataWriters* and *DataReaders* for *DomainParticipant*, publication, and subscription discovery traffic, while DPSE creates only one pair (for *DomainParticipant* discovery). This difference becomes more pronounced as you add more endpoints.

If your deployment topology is fixed and known at build time, use DPSE; it reduces both initialization overhead and discovery traffic. If your topology is dynamic or you need to discover endpoints at runtime, use DPDE. See *Discovery* for configuration details of both plugins.

### Transport and peer configuration

Enable only the transports your deployment requires. Each enabled transport adds initialization work: socket/port setup, internal tables, and resource allocation. See *Transport Registration* for details on how to register and enable transports.

By default, for the UDP transport, *Connext Micro* uses all available network interfaces. Use `allow_interface` or `deny_interface` on the transport's UDP_InterfaceFactoryProperty to restrict which network interfaces are used by a *DomainParticipant*. Limiting interfaces to those required by your deployment reduces the number of locators advertised in discovery announcements, the number of discovery messages sent, and the number of sockets opened at startup. See *UDP Transport* for details.

List only the minimal set of peers your *DomainParticipant* needs to discover in `DDS_DiscoveryQosPolicy.initial_peers`, and use explicit *DomainParticipant* indices (e.g., `[0-2]@_udp://10.10.30.1`) rather than the default range. If using multicast discovery, you can include just the multicast address instead of individual peers. Each entry in the peer list generates discovery announcements, so removing unnecessary entries and narrowing index ranges directly reduces discovery traffic. See *Configuring Participant Discovery Peers* for the peer descriptor format.

### QoS and resource configuration

The following subsections detail QoS and memory settings that can impact startup time, and how to optimize them.

### Memory and resource sizing

*Connext Micro* allocates resources up-front. Limits that are too small cause discovery failures (such as "out of records" or "resource limit exceeded"), while limits that are too large increase startup time and memory footprint. Size your resource limits to match your actual deployment.

- **Per-DomainParticipant discovery limits:** These limits control how many remote entities the *DomainParticipant* can store during discovery. If they are too small, you may see "out of records" or "resource limit exceeded" errors even when your per-endpoint limits are large enough.

  Configure the following fields under DDS_DomainParticipantQos.resource_limits:

| Field | Description |
|---|---|
| `remote_participant_allocation` | Maximum number of remote *DomainParticipants* expected in the DDS domain. |
| `remote_reader_allocation` | Maximum total number of remote *DataReaders* across all remote *DomainParticipants*. |
| `remote_writer_allocation` | Maximum total number of remote *DataWriters* across all remote *DomainParticipants*. |
| `matching_writer_reader_pair_allocation` | Maximum number of endpoint matches expected. A useful rule of thumb is `(local_reader_count + local_writer_count) * max_remote_participants`. |

- **Per-endpoint matching limits:** These limits control how many remote endpoints each local endpoint can track and match. Configure `max_remote_writers` under DDS_DataReaderQos.reader_resource_limits and `max_remote_readers` under DDS_DataWriterQos.writer_resource_limits.
- **Per-endpoint history and storage**: These limits affect how many samples are kept in memory, and can noticeably affect initialization time and memory footprint. Configure `max_samples`, `max_instances`, and `max_samples_per_instance` under DDS_DataReaderQos.resource_limits and DDS_DataWriterQos.resource_limits.

---

**Note:** For example, setting `remote_participant_allocation = 100` when your deployment has 5 *DomainParticipants* wastes memory and increases initialization time. Start with limits that match your known topology and increase them only if discovery fails.

---

#### Transport buffer sizing

Small send and receive buffers may limit discovery throughput and increase discovery completion time as *DomainParticipant* and endpoint counts grow. For the UDP transport, your operating system may cap effective socket buffer sizes regardless of what your application requests; even when you request large buffers, the OS may clamp them based on kernel limits. See *UDP Transport* for information on where UDP buffer settings are configured.

For the *Shared Memory Transport (SHMEM)*, the receive queue depth (`received_message_count_max`) and receive buffer size (`receive_buffer_size`) on NETIO_SHMEMInterfaceFactoryProperty control how much discovery traffic can be buffered. See *SHMEM Configuration* for details.

Adjust buffer sizes based on observed throughput, packet drops, and CPU utilization. If discovery traffic shows drops or "receive buffer errors," increase OS socket buffer limits and verify the effective buffer sizes.

### Discovery traffic tuning

As *DomainParticipant* and endpoint counts increase, discovery performance becomes bandwidth and burst-driven. The following properties control discovery traffic and timing. They are fields of DPDE_DiscoveryPluginProperty; see the API reference for the complete list.

- **Limit initial DomainParticipant announcement bursts:** When a *DomainParticipant* is enabled, it sends a burst of announcements. At larger scales, this burstiness can amplify congestion and increase discovery completion time. Configure `initial_participant_announcements` and `initial_participant_announcement_period` in DPDE_DiscoveryPluginProperty to control this behavior.

  For bandwidth-constrained links or large-scale deployments, reduce the announcement count or increase the interval from the default values.

- **Relax DomainParticipant liveliness traffic where permitted:** *DomainParticipants* liveliness assertions generate periodic network traffic. Aggressive settings add background load during discovery-heavy phases. Configure `participant_liveliness_assert_period` and `participant_liveliness_lease_duration` in DPDE_DiscoveryPluginProperty.

  Ensure `participant_liveliness_assert_period` is less than or equal to `participant_liveliness_lease_duration`. A common approach is to set the assert period to a fraction of the lease duration, to tolerate occasional assertion loss while still meeting liveliness requirements.

- **Use multicast discovery when available:** Unicast-only discovery increases bandwidth and processing cost as *DomainParticipant* count grows. If your deployment network supports multicast, use it to reduce fan-out.

  To enable multicast, add a multicast locator (e.g., `_udp://239.255.0.1`) to discovery.enabled_transports for discovery traffic and to user_traffic.enabled_transports for user data traffic. With DPSE, a multicast locator alone can be sufficient. With DPDE, always include both a unicast locator (e.g., `_udp://`) and a multicast locator; *Connext Micro* will select the appropriate locator based on the message type.

  For best-effort user data traffic, a multicast locator alone is sufficient. For reliable user data traffic, it is necessary to include both unicast and multicast locators to enable efficient communication.

- **Tune DPDE built-in endpoint reliability resources and timers:** DPDE's built-in endpoint buffering and retransmission settings can change discovery bandwidth and CPU usage, especially on lossy links or at large scale.

  Increasing `max_samples_per_builtin_endpoint_reader` can improve robustness (fewer drops and out-of-order rejections) during high discovery bursts, but doing so also increases memory usage.

  Smaller retransmission periods and higher retry limits can increase discovery traffic and CPU usage, but these may also reduce discovery completion time on lossy networks. The relevant DDS_DiscoveryPluginProperties fields are:

  – `builtin_writer_heartbeat_period`

- `builtin_writer_heartbeats_per_max_samples`
- `builtin_endpoint_reader_nack_period`
- `builtin_writer_max_heartbeat_retries`

When you tune these properties, document the tradeoff you are targeting (memory vs. drops vs. bandwidth) and validate by measuring discovery completion time alongside packet rates and CPU utilization.

#### Build settings

*Connext Micro* ships prebuilt release and debug libraries. Use the release libraries for startup-time measurements; debug libraries include additional error checking that increases initialization time.

Build your own application code with `-O2` (or equivalent) for production and for any measurements used to validate startup requirements. The same applies if you are building *Connext Micro* from source. Do not use `-O0` or `-Og` for startup-time measurements.

### 3.25.3 Measuring startup time

To optimize startup, you need to measure initialization time and discovery time separately.

---

**Note:** Console I/O during startup can account for a large portion of the total startup time and increase run-to-run variability, particularly on embedded targets. Defer or remove diagnostic output while measuring; if you need logs for debugging, capture them to a memory buffer rather than a serial console.

---

#### How to measure initialization time

Capture a timestamp before you create the *DomainParticipant* and another immediately after `DDS_Entity_enable()` returns for the *DomainParticipant* (or after the last entity is created, if autoenable is TRUE). The difference is your initialization time.

#### How to measure discovery time

Discovery time starts when the *DomainParticipant* is enabled and ends when your application has discovered all expected remote *DomainParticipants* and endpoints. You can detect discovery completion using listener callbacks or a *WaitSet*.

To use listener callbacks, set the `on_subscription_matched` callback on your *DataReaders* and the `on_publication_matched` callback on your *DataWriters*. Each time a callback fires with `status.current_count_change > 0`, a new match has occurred. When the total match count across all your endpoints reaches the expected value, discovery is complete. Capture a timestamp at that point.

To use a *WaitSet*, attach a `DDS_StatusCondition` with `DDS_SUBSCRIPTION_MATCHED_STATUS` (or `DDS_PUBLICATION_MATCHED_STATUS`) to a `DDS_WaitSet`. Call `DDS_WaitSet_wait()` in a loop, checking the match count after each trigger until all expected matches are reached.

For more information on listeners and status conditions, see *Receiving Data* and *Sending Data*.

#### What to record

Startup-time measurements are most useful when accompanied by the conditions under which they were taken. The following details help when comparing results across environments or configurations:

- Whether `autoenable_created_entities` was TRUE or FALSE;
- Whether release or debug libraries were used, and the compiler optimization level;
- The number of *DomainParticipants* and endpoints in the DDS domain during the measurement.

---

**Note:** Without these details, measurements from different environments or configurations cannot be meaningfully compared.

---

### 3.25.4 Validating startup time improvements

When evaluating a configuration change, compare initialization time and discovery time measurements before and after the change alongside the following indicators:

- **Transport buffer error counters:** check for receive buffer overflows or send failures. On Linux, `netstat -su` or `ss -su` shows UDP error counters. On QNX or other RTOS platforms, consult your OS documentation for equivalent counters.
- **Packet rates and discovery message counts:** use `tcpdump` or `Wireshark` to capture discovery traffic. Filter on the DDS discovery ports and count SPDP (*DomainParticipant*) and SEDP (endpoint) messages. High retransmission counts suggest packet loss or buffer exhaustion.
- **CPU utilization during discovery:** monitor CPU usage during the discovery phase. If a core is saturated, discovery processing may be the bottleneck rather than network I/O.

Reduced discovery completion time without increased packet loss or CPU saturation confirms the optimization is effective. If discovery time decreases but error counters rise, the change may have shifted the bottleneck rather than removed it.

# Chapter 4

# Transports

## 4.1 Introduction

In *RTI Connext Micro*, DDS entities exchange information using transports. Transports exhange data with peer transports, and *Connext Micro* entities can generally exchange information using different types of transports, e.g. UDPv4 or a serial port. All transports send and receive RTPS messages encapsulated in the transport's native format, e.g. UDP packets.

*RTI Connext Micro* has a pluggable-transports architecture. The core of *Connext Micro* is transport agnostic — it does not make any assumptions about the actual transports used to send and receive messages. Instead, *Connext Micro* uses an abstract "transport API" to interact with the transport plugins that implement that API. A transport plugin implements the abstract transport API, and performs the actual work of sending and receiving messages over a physical transport.

In *Connext Micro* a Network Input/Output (NETIO) interface is a software layer that may send and/or receive data from a higher and/or lower level locally, as well as communicate with a peer. A transport is a NETIO interface that is at the lowest level of the protocol stack. For example, the UDP NETIO interface is a transport.

A transport can send and receive on addresses as defined by the concrete transport. For example, the *Connext Micro* UDP transport can listen to and send to UDPv4 ports and addresses. In order to establish communication between two transports, the addresses that the transport can listen to must be determined and announced to other *DomainParticipants* that want to communicate with it. This document describes how the addresses are reserved and how these addresses are used by the DDS layer in *Connext Micro*.

While the NETIO interface is not limited to DDS, the rest of this document is written in the context of how *Connext Micro* uses the NETIO interfaces as part of the DDS implementation.

*Connext Micro* does not track the maximum message size of other nodes in the system. Therefore, *Connext Micro* relies on consistent configuration accross all the nodes in the system in order to successfully send and receive data. For example, if a *Connext Micro* node has a maximum message size of 8000 bytes and another *Connext Micro* node sends 9000 bytes the data will be sent, but not received.

## 4.2 Transport Registration

*RTI Connext Micro* supports different transports and transports must be registered with *RTI Connext Micro* before they can be used. A transport must be given a name when it is registered and this name is later used when configuring discovery and user-traffic. A transport name cannot exceed 7 UTF-8 characters.

The following example registers the UDP transport with *RTI Connext Micro* and makes it available to all DDS applications within the same memory space. Please note that each DDS applications creates its *own* instance of a transport. Resources are *not* shared between instances of a transport unless stated.

For example, to register two UDP transports with the names myudp1 and myudp2, the following code is required:

```
DDS_DomainParticipantFactory *factory;
RT_Registry_T *registry;
struct UDP_InterfaceFactoryProperty udp_property;

factory = DDS_DomainParticipantFactory_get_instance();
registry = DDS_DomainParticipantFactory_get_registry(factory);

/* Set UDP properties */
if (!RT_Registry_register(registry,"myudp1",
                          UDP_InterfaceFactory_get_interface(),
                          &udp_property._parent._parent,NULL))
{
    return error;
}

/* Set UDP properties */
if (!RT_Registry_register(registry,"myudp2",
                          UDP_InterfaceFactory_get_interface(),
                          &udp_property._parent._parent,NULL))
{
    return error;
}
```

Before a DomainParticipant can make use of a registered transport, it must enable it for use within the DomainParticipant. This is done by setting the TransportQoS. For example, to enable only myudp1, the following code is required (error checking is not shown for clarity):

```
DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,1);
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                                    REDA_String_dup("myudp1");
```

To enable both transports:

```
DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,2);
DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,2);
```

(continues on next page)

(continued from previous page)

```
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                                    REDA_String_dup("myudp1");
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                                    REDA_String_dup("myudp2");
```

Before enabled transports may be used for communication in *Connext Micro*, they must be registered and added to the DiscoveryQos and UserTrafficQos policies. Please see the section on *Discovery* for details.

## 4.3 Transport Addresses

In order for DDS entities to communicate, the DDS entities must know each other's location. DDS entities may be colocated in the same *DomainParticipant*, may be located in different *DomainParticipants* within the same node, or may be located on different nodes connected by a network.

In DDS, a location is called a *locator*. A locator uniquely describes how to reach one or more DDS entitites in a network. A DDS locator consists of the following parts:

- The locator *kind* identifies the type of locators; e.g., UDPv4.
- The locator *port* identifies the location of DDS entities at an address. The port number of a locator is not directly configurable; rather, it is configured indirectly by the `rtps_well_known_ports` attribute of the WireProtocolQosPolicy of the *DomainParticipant*'s QoS, where a well-known, interoperable RTPS port number is assigned.
- The locator *address* indentifies the network address. Transports are concerned with exchanging messages using the network address.

### 4.3.1 Reserving addresses and ports

Address reservation is the process to determine which locators should be used in the discovery announcement. Which transports and addresses to be used are determined as described in *Discovery*.

When a *DomainParticipant* is created, it calculates a port number and tries to reserve this port on all addresses available in *all* the transports based on the registration properties. If the port cannot be reserved on all transports, then it release the port on *all* transports and tries again. If no free port can be found the process fails and the *DomainParticipant* cannot be created.

**Warning:** If an address is specified without the transport name as a prefix, (e.g. "192.168.1.1" instead of "_udp://192.168.1.1") and multiple transports understand the address, the *first* transport that understands the address will also try to reserve it. Transports are sorted alphabetically by name. This feature allows backward compatibility with earlier versions of *Connext Micro*, but *should not be used*; it may be deprecated in future versions. Always specify addresses using the transport name as the prefix.

### 4.3.2 Address limitations

The number of locators which can be announced is limited to *only* the first *four* for each type across *all* transports available for each policy. If more than four are available of any kind, these are *ignored*. This is by design, although it may be changed in the future. The order in which the locators are read from the platform is also not known; thus, if there are more than four possible locators, the four locators which will be used are not deterministic.

To ensure that *all* the desired addresses and *only* the desired address are used in a transport, follow these rules:

- Make sure that no more than four unicast addresses and four multicast addresses can be returned across *all* transports for discovery traffic.
- Make sure that no more than four unicast addresses and four multicast addresses can be returned across *all* transports for user traffic.
- Make sure that no more than four unicast addresses and four multicast addresses can be returned across *all* transports for user-traffic, for *DataReader* and *DataWriter* specific locators, and that they do *not* duplicate any of the *DomainParticipant*'s locators.

### 4.3.3 Address notation

In *Connext Micro*, all addresses are specified as ASCII strings. The full address format is:

```
[ ] denotes range or discreet values, unless enclosed in ''
    which means a literal.

ADDRESS = PREFIX://<ADDRESS> |
          @PREFIX://<ADDRESS> |
          INDEX@PREFIX://<ADDRESS>

INDEX = INTEGER | '[' INTEGER ']' | '[' INTEGER-INTEGER ']' | '[' -INTEGER ']'

PREFIX = [a-zA-Z_][0-9a-zA-Z_]+

INTEGER = DEC_INTEGER | HEX_INTEGER

DEC_INTEGER = [0-9]+

HEX_INTEGER = [0x|0X][0-9a-fA-F]+

ADDRESS = 0 or more 8bit characters
```

**Warning:** Note that while a locator without a `PREFIX` is supported, it *should* always be used and is thus *not* marked as optional in the grammar.

## 4.4 Transport Port Number

The port number of a locator is not directly configurable. Rather, it is configured indirectly by the DDS_WireProtocolQosPolicy (rtps_well_known_ports) of the *DomainParticipant's* QoS, where a well-known, interoperable RTPS port number is assigned.

## 4.5 Interoperating with Connext Professional Transports

When interoperating with *Connext Professional*, *Connext Micro* must specify at least one unicast transport for each *DataWriter* and *DataReader*, either from DDS_DomainParticipantQos::transports or the endpoint DDS_DataReaderQos::transport and DDS_DataWriterQos::transport, as it expects to use the unicast transport's RTPS port mapping to determine automatic participant IDs if needed. This also affects *Connext Micro* itself, where participant IDs must be set manually if only multicast transports are enabled.

Also, when interoperating with *Connext Professional*, only one multicast transport can be specified per *DataReader* of *Connext Micro*.

## 4.6 RTPS

The RTPS transport encapsulates user-data in RTPS messages and parses received RTPS messages for user-data. This chapter describes how to configure RTPS.

### 4.6.1 Registration of RTPS

RTPS is automatically registered when a *DDS_DomainParticipantFactory* is initialized with *DDS_DomainParticipantFactory_get_instance()*. In order to change the RTPS configuration, it is necessary to first unregister it from the participant factory, set the properties, and then register RTPS with the new properties. This process is identical to other plugins in *Connext Micro*, such as the UDP transport and discovery plugins.

The following code shows the steps:

```
int main(int argc, char *argv[])
{
    struct RTPS_InterfaceFactoryProperty *rtps_property = NULL;
    DDS_DomainParticipantFactory *factory = NULL;
    RT_Registry_T *registry = NULL;
    struct DDS_DomainParticipantQos dp_qos =
                            DDS_DomainParticipantQos_INITIALIZER;
    DDS_DomainId_t domain_id = 1;

    /* get the Domain Participant factory and registry*/
    factory = DDS_DomainParticipantFactory_get_instance();

    registry =  DDS_DomainParticipantFactory_get_registry(factory);
```

(continues on next page)

(continued from previous page)

```
    /* unregister the RTPS transport */
    if (!RT_Registry_unregister(registry, NETIO_DEFAULT_RTPS_NAME,
                                NULL,NULL))
    {
        printf("failed to unregister rtps\n");
        return 0;
    }

    rtps_property = (struct RTPS_InterfaceFactoryProperty *)
            malloc(sizeof(struct RTPS_InterfaceFactoryProperty));

    if (rtps_property == NULL)
    {
        printf("failed to allocate rtps properties\n");
        return 0;
    }

    /* Set the new properties and register RTPS again */

    if (!RT_Registry_register(registry, NETIO_DEFAULT_RTPS_NAME,
                RTPS_InterfaceFactory_get_interface(),
                (struct RT_ComponentFactoryProperty*)rtps_property,
                 NULL))
    {
        printf("failed to register rtps\n");
        return 0;
    }


    DDS_DomainParticipantFactory_create_participant(
       factory, domain_id,&dp_qos, NULL,DDS_STATUS_MASK_NONE);
}
```

Please note that the RTPS properties *must* be valid for the *entire* life-cycle of the participant factory because RTPS *does not* make an internal copy. This saves memory when properties are stored in preallocated memory (for example in ROM).

### 4.6.2 Overriding the Builtin RTPS Checksum Functions

Some applications may require specialized functions to guarantee message integrity or may have special hardware that supports faster checksum calculations. *Connext Micro* provides a way for users to override the builtin checksum functions. Note that if a different checksum is calculated it may prevent interoperability with other DDS implementations.

#### Checksum function definition

A checksum function must define a structure of the following type:

```
typedef struct RTPS_ChecksumClass
{
    RTPS_ChecksumClassId_T class_id;
    void *context;
    RTPS_ChecksumCalculate_T checksum_calculate;
} RTPS_ChecksumClass_T;
```

The type has three members:

1. class_id - The class ID must be:
   - RTPS_CHECKSUM_CLASSID_BUILTIN32 for the 32-bit checksum.
   - RTPS_CHECKSUM_CLASSID_BUILTIN64 for the 64-bit checksum.
   - RTPS_CHECKSUM_CLASSID_BUILTIN128 for the 128-bit checksum.
2. context - An opaque object for you to provide context for this function. This context will be passed to the *checksum_calculate* every time it is called.
3. checksum_calculate - The function pointer to the checksum function. The function is defined as

```
typedef RTI_BOOL
(*RTPS_ChecksumCalculate_T)(void *context,
                            const struct REDA_Buffer *buf,
                            RTI_UINT32 buf_length,
                            RTPS_Checksum_T *checksum);
```

   - context: *Connext Micro* will pass in the context as defined in the class.
   - buf: An array of REDA_Buffer. Each REDA_Buffer includes a pointer and size of the buffer.
   - buf_length: The size of the array.
   - checksum: This is the out parameter of this function. It is a union defined as follows:

```
typedef union RTPS_Checksum
{
        RTI_UINT32 checksum32;
```

(continues on next page)

(continued from previous page)

```
        RTI_UINT64 checksum64;
        RTI_UINT8 checksum128[16];
} RTPS_Checksum_T;
```

Please note the following *important* information regarding the output values:

1. The number returned in checksum32 is assumed to be in *host order* endianness.
2. The number returned in checksum64 is assumed to be in *host order* endianness.
3. checksum128 is treated as an octet array.

### 4.6.3 Example

Below is an example implementation of a custom CRC-32 function using the Intel intrinsic functions. It shows the QoS that needs to be set, as well as how to override the builtin checksum function.

```
RTI_BOOL
CrcClassTest_custom_crc32_other(void *context,
                                const struct REDA_Buffer *buf,
                                RTI_UINT32 buf_length,
                                RTPS_Checksum_T *checksum)
{
    RTI_UINT32 crc = 0;
    RTI_UINT8 *data = (RTI_UINT8 *) buf[0].pointer;
    RTI_UINT32 length = buf[0].length;
    RTI_INT32 k;

    UNUSED_ARG(context);
    UNUSED_ARG(buf_length);

    for (k = 0; k < length; k++)
    {
        crc = _mm_crc32_u8(crc, data[k]);
    }

    checksum->checksum32 = crc;

    return RTI_TRUE;
}


int main(int argc, char *argv[])
{
    struct DDS_DomainParticipantQos dp_qos =
            DDS_DomainParticipantQos_INITIALIZER;
    struct RTPS_InterfaceFactoryProperty *rtps_property = NULL;
    DDS_DomainParticipantFactory *factory = NULL;
    RT_Registry_T *registry = NULL;
    DDS_DomainId_t domain_id = 1;
```

(continues on next page)

(continued from previous page)

```
    /* Instantiate a RTPS_ChecksumClass for your custom function*/
    struct RTPS_ChecksumClass custom_crc32 =
    {
        RTPS_CHECKSUM_CLASSID_BUILTIN32, /*class_id*/
        NULL, /*context*/
        CrcClassTest_custom_crc32_other /*Custom function*/
    };

    /* get the Domain Participant factory and registry*/
    factory = DDS_DomainParticipantFactory_get_instance();

    registry =  DDS_DomainParticipantFactory_get_registry(factory);


    /* unregister the RTPS transport */
    if (!RT_Registry_unregister(registry, NETIO_DEFAULT_RTPS_NAME,
                                NULL,NULL))
    {
        printf("failed to unregister rtps\n");
        return 0;
    }

    rtps_property = (struct RTPS_InterfaceFactoryProperty *)
            malloc(sizeof(struct RTPS_InterfaceFactoryProperty));

    if (rtps_property == NULL)
    {
        printf("failed to allocate rtps properties\n");
        return 0;
    }


    /* the rtps property takes the structure with the custom
     * function
     */

    *rtps_property = RTPS_INTERFACE_FACTORY_DEFAULT;
    rtps_property->checksum.allow_builtin_override = RTI_TRUE;
    rtps_property->checksum.builtin_checksum32_class = custom_crc32;

    /* register the RTPS transport */
    if (!RT_Registry_register(registry, NETIO_DEFAULT_RTPS_NAME,
                RTPS_InterfaceFactory_get_interface(),
                (struct RT_ComponentFactoryProperty*)rtps_property,
                 NULL))
    {
        printf("failed to register rtps\n");
        return 0;
    }
```

(continues on next page)

(continued from previous page)

```
    /* modify the domain participant qos */
    dp_qos.protocol.compute_crc = DDS_BOOLEAN_TRUE;
    dp_qos.protocol.check_crc = DDS_BOOLEAN_TRUE;
    dp_qos.protocol.require_crc = DDS_BOOLEAN_TRUE;
    dp_qos.protocol.computed_crc_kind =  DDS_CHECKSUM_BUILTIN32;
    dp_qos.protocol.allowed_crc_mask = DDS_CHECKSUM_BUILTIN32;

    /* use the qos and the factory to create a participant */

    DDS_DomainParticipantFactory_create_participant(
       factory, domain_id,&dp_qos, NULL,DDS_STATUS_MASK_NONE);
}
```

## 4.7 INTRA Transport

The builtin intra participant transport (INTRA) is a transport that bypasses RTPS and reduces the number of data-copies from three to one for data published by a *DataWriter* to a *DataReader* within the same participant. When a sample is published, it is copied directly to the data reader's cache (if there is space). This transport is used for communication between *DataReaders* and *DataWriters* created within the same participant by default.

Please refer to *Threading Model* for important details regarding application constraints when using this transport.

### 4.7.1 Registering the INTRA Transport

The builtin INTRA transport is a *RTI Connext Micro* component that is automatically registered when the DomainParticipantFactory_get_instance() method is called. By default, data published by a *DataWriter* is sent to all *DataReaders* within the same participant using the INTRA transport.

In order to prevent the INTRA transport from being used it is necessary to remove it as a transport and a user-data transport prior to creating the participant. The following code shows how to only use the builtin UDP transport for user-data.

```
struct DDS_DomainParticipantQos dp_qos =
                            DDS_DomainParticipantQos_INITIALIZER;

REDA_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,1);
REDA_StringSeq_set_length(&dp_qos.transports.enabled_transports,1);
*REDA_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                    REDA_String_dup(NETIO_DEFAULT_UDP_NAME);

/* Use only unicast for user-data traffic. */
REDA_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports,1);
REDA_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports,1);
*REDA_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports,0) =
                                    REDA_String_dup("_udp://");
```

Note that the INTRA transport is never used for discovery traffic internally. It is not possible to disable matching of *DataReaders* and *DataWriters* within the same participant.

### 4.7.2 Configuring QoS policies

Most configurations of the Reliability and Durability QoS policies will function as intended over the INTRA transport. However, because the INTRA transport bypasses RTPS reliability and DDS durability queues, your application may lose data when using certain settings, as described below.

#### Reliability

If a *DataWriter* configured with `RELIABLE` Reliability and `KEEP_ALL` History matches with any *DataReader* via the INTRA transport, the *DataWriter* will eventually run out of resources in its cache. This occurs because the *DataReader* has no path to send sample acknowledgements (ACKNACKS) to the *DataWriter* over INTRA. Once the *DataWriter*'s queue fills up, it will stop sending data. Therefore, this configuration should *not* be used when communicating via the INTRA transport.

All other configurations of Reliability and History are supported, including a `RELIABLE` *DataWriter* with `KEEP_LAST` History.

#### Durability

The `TRANSIENT_LOCAL` Durability setting directs a *DataWriter* to keep samples in its queue for late-joining *DataReaders*, but it has no path to deliver them when communicating via INTRA. Late-joining `TRANSIENT_LOCAL` *DataReaders* will not receive historical data as intended.

---

**Note:** `TRANSIENT_LOCAL` Durability does not prevent *DataWriters* and *DataReaders* from communicating over INTRA after both are created; only the historical samples in the *DataWriter*'s queue will fail to be delivered.

---

However, if you create all *DataReaders* before creating any *DataWriters*, `TRANSIENT_LOCAL` Durability is not required because the *DataReaders* will start receiving data immediately from any matched *DataWriters* without missing any samples.

### 4.7.3 Threading Model

The INTRA transport does not create any threads. Instead, a *DataReader* receives data over the INTRA transport in the context of the *DataWriter*'s *send thread*.

This model has two *important limitations*:

- Because a *DataReader*'s on_data_available() listener is called in the context of the *DataWriter*'s send thread, a *DataReader* may potentially process data at a different priority than intended (the *DataWriter*'s). While it is generally not recommended to process

data in a *DataReader*'s on_data_available() listener, it is particularly important *to not do so* when using the INTRA transport. Instead, use a DDS WaitSet or a similar construct to wake up a separate thread to process data.

- Because a *DataReader*'s on_data_available() listener is called in the context of the *DataWriter*'s send thread, any method called in the on_data_available() listener is done in the context of the *DataWriter*'s stack. Calling a *DataWriter* **write()** in the callback could result in an infinite call stack. Thus, it is recommended *not* to call in this listener any *Connext Micro* APIs that write data.

## 4.8 Shared Memory Transport (SHMEM)

This section describes the optional builtin *RTI Connext Micro* SHMEM transport and how to configure it.

Shared Memory Transport (SHMEM) is an optional transport that can be used in *Connext Micro*. It is part of a standalone library that can be optionally linked in.

The SHMEM Transport also allows *Connext Micro* to transmit data samples without copying them internally. For an overview of this feature, see *Zero Copy Transfer*.

Currently, *Connext Micro* supports the following functionality:

- Unicast
- Configuration of the shared memory receive queues

### 4.8.1 Registering the SHMEM Transport

The builtin SHMEM transport is a *Connext Micro* component that needs to be registered before a *DomainParticipant* can be created with the ability to send data across shared memory. Unlike the UDP Transport, this transport is not automatically registered. Register the transport using the code snippet below:

```
#include "netio_shmem/netio_shmem.h"

...

{
    DDS_DomainParticipantFactory *factory = NULL;
    RT_Registry_T *registry = NULL;
    struct NETIO_SHMEMInterfaceFactoryProperty shmem_property = NETIO_
↪SHMEMInterfaceFactoryProperty_INITIALIZER;
    struct DDS_DomainParticipantQos dp_qos = DDS_DomainParticipantQos_INITIALIZER;

    /* Optionally configure the transport settings */
    shmem_property.received_message_count_max = ...
    shmem_property.receive_buffer_size = ...
    shmem_property.message_size_max = ...
```

(continues on next page)

(continued from previous page)

```
    factory = DDS_DomainParticipantFactory_get_instance();

    registry = DDS_DomainParticipantFactory_get_registry(factory);
    if (!RT_Registry_register(
            registry,
            "_shmem",
            NETIO_SHMEMInterfaceFactory_get_interface(),
            (struct RT_ComponentFactoryProperty*)&shmem_property,
            NULL))
    {
        /* ERROR */
    }

    /* Enable the transport on a Domain Participant */
    DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,1);
    DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,1);
    *DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) = DDS_String_
↪dup("_shmem");

    DDS_StringSeq_set_maximum(&dp_qos.discovery.enabled_transports,1);
    DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports,1);
    *DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports,0) = DDS_String_
↪dup("_shmem://");

    DDS_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports,1);
    DDS_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports,1);
    *DDS_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports,0) = DDS_String_
↪dup("_shmem://");

    DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers,1);
    DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers,1);
    *DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers,0) = DDS_String_dup("_
↪shmem://");

    ...

    /* Explicitly unregister the shared memory transport before clean up */
    if (!RT_Registry_unregister(
            registry,
            "_shmem",
            NULL,
            NULL)
    {
        /* ERROR */
    }
}
```

**Note:** `dp_qos.transports.enabled_transports` refers to a string that contains only a registered transport name, such as `_shmem`. `dp_qos.discovery.enabled_transports` and `dp_qos.user_traffic.enabled_transports` require a transport name prefix and (optional) address suffix, as described in *Configuring Discovery Data Reception* and *Configuring User Data Reception*.

`dp_qos.discovery.initial_peers` also includes an optional index prefix, as described in *Configuring Participant Discovery Peers*.

The above snippet will register a transport with the default settings. To configure it, change the invidiual configurations as described in *SHMEM Configuration*.

When a component is registered, the registration takes the properties and a listener as the 3rd and 4th parameters. The registration of the shared memory component will make a copy of the properties configurable within a shared memory transport. There is currently no support for passing in a listener as the 4th parameter.

It should be noted that the SHMEM transport can be registered with any name, but all transport QoS policies and initial peers must refer to this name. If a transport is referred to and it does not exist, an error message is logged.

While it is possible to register multiple SHMEM transports, it is not possible to use multiple SHMEM transports within the same participant. The reason is that SHMEM port allocation is not synchronized between transports.

### 4.8.2 Threading Model

The SHMEM transport creates one receive thread for each unique SHMEM receive address and port. Thus, by default two SHMEM threads are created:

- A unicast receive thread for discovery data
- A unicast receive thread for user data

Each receive thread will create a shared memory segment that will act as a message queue. Other *DomainParticipants* will send RTPS message to this message queue.

This message queue has a fixed size and can accommodate a fixed number of messages (*received_message_count_max*) each with a maximum payload size of (*message_size_max*). The total size of the queue is configurable with (*receive_buffer_size*).

#### Configuring SHMEM Receive Threads

All threads in the SHMEM transport share the same thread settings. It is important to note that all the SHMEM properties must be set before the SHMEM transport is registered. *Connext Micro* preregisters the SHMEM transport with default settings when the DomainParticipantFactory is initialized. To change the SHMEM thread settings, use the following code.

```
struct SHMEM_InterfaceFactoryProperty shmem_property = NETIO_SHMEMInterfaceFactoryProperty_
↪INITIALIZER

shmem_property.recv_thread_property.options = ...;

/* The stack-size is platform dependent, it is passed directly to the OS */
shmem_property.recv_thread_property.stack_size = ...;
```

(continues on next page)

(continued from previous page)

```
/* The priority is platform dependent, it is passed directly to the OS */
shmem_property.recv_thread_property.priority = ...;

if (!RT_Registry_register(registry, "_shmem",
                          SHMEM_InterfaceFactory_get_interface(),
                          (struct RT_ComponentFactoryProperty*)&shmem_property,
                          NULL))
{
    /* ERROR */
}
```

### 4.8.3 SHMEM Configuration

All the configuration of the SHMEM transport is done via the struct SHMEM_InterfaceFactoryProperty structure:

```
struct NETIO_SHMEMInterfaceFactoryProperty
{
    struct NETIO_InterfaceFactoryProperty _parent;
    /* Max number of received message sizes that can be residing
       inside the shared memory transport concurrent queue
     */
    RTI_INT32 received_message_count_max;
    /* The size of the receive socket buffer */
    RTI_INT32 receive_buffer_size;
    /* The maximum size of the message which can be received */
    RTI_INT32 message_size_max;
    /* Thread properties for each receive thread created by this
       NETIO interface.
     */
    struct OSAPI_ThreadProperty recv_thread_property;
};
```

#### received_message_count_max

The number of maximum RTPS messages that can be inside a receive thread's receive buffer. By default this is 64.

#### receive_buffer_size

The size of the message queue residing inside a shared memory region accessible from different processes. The default size is ((*received_message_count_max* * *message_size_max*) / 4).

#### message_size_max

The size of an RTPS message that can be sent across the shared memory transport. By default this number is 65536.

#### recv_thread_property

The recv_thread field is used to configure all the receive threads. Please refer to *Threading Model* for details.

#### pro_minimum_compatiblity_version

The minimum version of *Connext Professional* with which to guarantee compatibility when using shared memory. This only needs to be specified if `dds.transport.minimum_compatibility_version` has been specified in *Connext Professional* and compatibility with *Connext Micro* is required. The default value is `DDS_PRODUCTVERSION_UNKNOWN`.

See Capturing Shared Memory Traffic in the Core Libraries User's Manual for more information on `dds.transport.minimum_compatibility_version` in *Connext Professional*.

### 4.8.4 Caveats

#### Leftover shared memory resources

*Connext Micro* implements the shared memory transport and utilizes shared memory semaphores that can be used concurrently by processes. *Connext Micro* implements a shared memory mutex from a shared memory semaphore. If an application exits ungracefully, then the shared memory mutex may be left in a state that prevents it from being used. This can occur because the *Connext Micro* Shared Memory Transport tries to re-use and clean up and leftover segments as a result of an applications ungraceful termination. If ungraceful termination occurs, the leftover shared memory mutexes need to be cleaned up either manually or by restarting the system.

The same applies to shared memory semaphores. If an application exists ungracefully, there can be leftover shared memory segments.

#### Darwin and Linux systems

In the case of Darwin and Linux systems which use SysV semaphores, you can view any leftover shared memory segments using **ipcs -a**. They can be removed using the **ipcrm** command. Shared memory keys used by *Connext Micro* are in the range of 0x00400000. For example:

- `ipcs -m | grep 0x004`

The shared semaphore keys used by *Connext Micro* are in the range of 0x800000; the shared memory mutex keys are in the range of 0xb00000. For example:

- `ipcs -m | grep 0x008`

- `ipcs -m | grep 0x00b`

#### QNX systems

QNX® systems use POSIX® APIs to create shared memory segments or semaphores. The shared memory segment resources are located in **/dev/shmem** and the shared memory mutex and semaphores are located in **/dev/sem**.

To view any leftover shared memory segments when no *Connext Micro* applications are running:

- `ls /dev/shmem/RTIOsapi*`
- `ls /dev/sem/RTIOsapi*`

To clean up the shared memory resources, remove the files listed.

#### Windows and VxWorks systems

On Windows and VxWorks® systems, once all the processes that are attached to a shared memory segment, shared memory mutex, or shared memory semaphores are terminated (either gracefully or ungracefully), the shared memory resources will be automatically cleaned up by the operating system.

## 4.9 Zero Copy v2 Transport

The Zero Copy v2 transport enables *RTI Connext Micro* to share data samples between publishers and subscribers without serializing, transmitting, or deserializing the samples. For an overview of this feature and its utility, see *Zero Copy Transfer*.

This section outlines the basic steps required to enable the Zero Copy v2 transport in an application. All the example code shown below is taken from a Zero Copy v2 application that you can generate using *rtiddsgen* (see *Example Applications* for more details).

### 4.9.1 Generate example and type support files

First, identify types that require Zero Copy transfer and annotate them with the `@transfer_mode(SHMEM_REF)` annotation. See the example IDL file below:

```
@transfer_mode(SHMEM_REF)
struct HelloWorld {
    @key long  id;
    long  data[100];
};
```

*rtiddsgen* generates additional TypePlugin code when a type is annotated with `@transfer_mode(SHMEM_REF)` in the IDL files. This code allows a *DataWriter* and a *DataReader* to communicate using a reference to the sample in shared memory.

**Note:** Zero Copy v2 for *Connext Micro* only supports contiguous data types; this means that fixed-size arrays are supported, but sequences and strings are not.

Next, generate the type support and example files with the following command:

```
rtiddsgen -micro -example -exampleTemplate zcv2 -language C HelloWorld.idl
```

The generated files will appear in the same directory as the type file.

### 4.9.2 Initialize the Zero Copy v2 transport

Before the Zero Copy v2 transport can be used, it must be initialized. This must be done before creating a *DomainParticipant* and after registering the *DataReader* and *DataWriter* history plugins. The order is important because the Zero Copy v2 transport will perform actions on the history components during initialization.

The following example code from **HelloWorldApplication.c** demonstrates how to initialize the Zero Copy v2 transport with NDDS_Transport_ZeroCopy_initialize():

```
if (!NDDS_Transport_ZeroCopy_initialize(registry, NULL, NULL))
{
    printf("failed to initialize zero copy\n");
    /* handle error */
}
```

### 4.9.3 Register the Zero Copy v2 transport

The Zero Copy v2 transport needs a notification mechanism to notify *DataReaders* when a *DataWriter* has published data samples. RTI provides a default notification mechanism based on POSIX. You can also implement your own custom notification mechanism, but doing so is beyond the scope of this documentation; for more information, contact support@rti.com.

The default provided by RTI is a POSIX implementation of the notification mechanism. This mechanism is based on a monitor implemented in shared memory. In this documentation, we will assume that you are using the default implementation unless otherwise noted.

Once the Zero Copy transport is initialized, configure the `notif` interface factory with the ZCOPY_NotifInterfaceFactoryProperty property. This property has three fields you need to set:

- max_samples_per_notif: The number of samples processed per notification. By default this value is 1. Note that a high value may starve other threads from progressing.
- user_intf: This is the implementation of your chosen notification mechanism. It is populated automatically if you are using the default implementation.
- user_property: Any properties associated with your chosen notification mechanism. *Connext Micro* treats this as an opaque pointer.

When using the default mechanism provided by RTI, the user_property mentioned above is resolved to ZCOPY_NotifMechanismProperty. Both of these properties are required to configure the transport.

See the following example from **HelloWorldApplication.c**:

```
struct ZCOPY_NotifInterfaceFactoryProperty  notif_prop;
struct ZCOPY_NotifMechanismProperty notif_mech_prop;
notif_mech_prop.intf_addr = 0;
notif_prop.user_property = &notif_mech_prop;
notif_prop.max_samples_per_notif = 1;
```

For more information on these properties, see the *Configuration* section.

Finally, call ZCOPY_NotifMechanism_register() (a utility function on the default notification mechanism) to register the Zero Copy v2 transport with the default notification mechanism. This makes it available for use. The following example registers a `notif` with the name `NETIO_DEFAULT_NOTIF_NAME`:

```
if (!ZCOPY_NotifMechanism_register(registry, NETIO_DEFAULT_NOTIF_NAME, &notif_prop))
{
    printf("failed to register notif\n");
    goto done;
}
```

### 4.9.4 Enable transports

With the specific notification mechanism registered, you can enable the Zero Copy and UDP transports for the *DomainParticipant*. Consider the following example code:

```
if (!DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports, 2))
{
    printf("failed to set transports.enabled_transports maximum\n");
    goto done;
}
if (!DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports, 2))
{
    printf("failed to set transports.enabled_transports length\n");
    goto done;
}
/* UDP and Notif are enabled*/
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports, 0) =
        DDS_String_dup(NETIO_DEFAULT_NOTIF_NAME);
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports, 1) =
        DDS_String_dup(NETIO_DEFAULT_UDP_NAME);

/* Discovery takes place over UDP */
DDS_StringSeq_set_maximum(&dp_qos.discovery.enabled_transports, 1);
DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports, 1);
*DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports, 0) =
        DDS_String_dup("_udp://");
```

(continues on next page)

(continued from previous page)

```
/* User data uses Notif */
DDS_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports, 1);
DDS_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports, 1);
*DDS_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports, 0) =
        DDS_String_dup("notif://");
```

**Note:** The *UDP Transport* or *Shared Memory Transport (SHMEM)* must be registered while using Zero Copy v2 transfer because DDS Discovery requires one of them in order to function (see *Discovery* for more details).

### 4.9.5 Sample management

When using the Zero Copy v2 transport, each *DataWriter* manages a pool of samples, and the application must obtain samples from this pool using get_loan(). We can see this in the following example:

```
hw_datawriter = HelloWorldDataWriter_narrow(datawriter);
retcode = HelloWorldDataWriter_get_loan(hw_datawriter, &sample);
if (retcode != DDS_RETCODE_OK)
{
    printf("ERROR: Failed to loan sample\n");
}
retcode = HelloWorldDataWriter_write(hw_datawriter, sample, &DDS_HANDLE_NIL);
if (retcode != DDS_RETCODE_OK)
{
    printf("ERROR: Failed to write to sample\n");
}
```

As seen above, the *DataWriter* **must** get a loan before each write call; it cannot write a loaned sample multiple times. The *DataWriter* does not need to explicitly return any loan to the pool, since this is managed by the middleware. However, if a loaned sample will not be written, it can be discarded with discard_loan().

**Warning:** It is not possible to write a sample that has not been obtained with get_loan().

**Note:** A Zero Copy-enabled *DataWriter* can also send samples using other transports (such as UDPv4) to non-Zero Copy *DataReaders*. When a *DataWriter* uses both the Zero Copy v2 transport and a transport which uses serialized data (such as UDP), the same sample is sent over all transports.

This may adversely affect performance, since the sample must be serialized for network transmission even if it is in shared memory. For best performance, you should consider an architecture where

a *DataWriter* matches either with Zero Copy-enabled or non Zero Copy-enabled *DataReaders*, but not both.

---

On the *DataReader* side, Zero Copy v2 application code is identical to subscribing applications not using Zero Copy.

When a *DataReader* calls read() or take() and receives samples, it is being given samples that are loaned from the *DataWriter*'s pool. Thus, failing to return the loan when the sample is no longer needed will deplete the available samples in the pool, eventually causing calls to get_loan() to fail.

### 4.9.6 Limit type plugin heap creation

By default, when a participant registers a transport that requires serialization, the middleware allocates serialization buffers so that *DataWriters* can send to *DataReaders* over the interface. In a Zero Copy v2 deployment where the *DataReaders* should **only** receive over the Zero Copy v2 transport and never fall back to CDR delivery, this plugin creation is unnecessary and can be suppressed. This has the additional benefit of preventing the middleware from creating a memory pool for the XCDR1 or XCDR2 plugin and its serialization buffers, which can reduce memory requirements.

---

**Note:** In a system where a *DataWriter* is expected to match multiple readers with varying transports, these Qos should not be modified and the allocation of serialization buffers should not be suppressed. This is only when all readers will only use Zero Copy v2.

---

To suppress the creation of the XCDR serialization buffers, set the TransportEncapsulationQosPolicy on the *DataWriter* to contain exactly one TransportEncapsulationSettings_t entry whose `encapsulations` sequence holds only DDS_ENCAPSULATION_ID_SHMEM_V2. When the middleware sees that the writer's sole offered encapsulation is `DDS_ENCAPSULATION_ID_SHMEM_V2`, it skips the creation of any XCDR1 or XCDR2 memory plugin for the participant's other transports (such as UDP).

The DataRepresentationQosPolicy must be set consistently across the system. Pair `DDS_ENCAPSULATION_ID_SHMEM_V2` with the correct data representation, either `DDS_XCDR_DATA_REPRESENTATION` or `DDS_XCDR2_DATA_REPRESENTATION` to ensure the correct wire format is negotiated during discovery.

```
struct DDS_TransportEncapsulationSettings_t *encap_settings = NULL;

/* Add exactly one TransportEncapsulationSettings_t entry */
DDS_TransportEncapsulationSettingsSeq_set_maximum(
        &dw_qos.encapsulation.value, 1);
DDS_TransportEncapsulationSettingsSeq_set_length(
        &dw_qos.encapsulation.value, 1);

encap_settings = DDS_TransportEncapsulationSettingsSeq_get_reference(
        &dw_qos.encapsulation.value, 0);
```

(continues on next page)

(continued from previous page)

```
/* Restrict the encapsulation to SHMEM_V2 only */
DDS_EncapsulationIdSeq_set_maximum(
        &encap_settings->encapsulations, 1);
DDS_EncapsulationIdSeq_set_length(
        &encap_settings->encapsulations, 1);
*DDS_EncapsulationIdSeq_get_reference(
        &encap_settings->encapsulations, 0) =
        DDS_ENCAPSULATION_ID_SHMEM_V2;

/* Set the matching representation */
DDS_DataRepresentationIdSeq_set_maximum(
        &dw_qos.representation.value, 1);
DDS_DataRepresentationIdSeq_set_length(
        &dw_qos.representation.value, 1);
*DDS_DataRepresentationIdSeq_get_reference(
        &dw_qos.representation.value, 0) =
        DDS_XCDR2_DATA_REPRESENTATION;
```

**Note:** When `DDS_ENCAPSULATION_ID_SHMEM_V2` is the only offered encapsulation, any *DataReader* whose transport has no CDR plugin is forced to use the notification transport to communicate back to the writer even if that *DataReader* also has UDP enabled. To allow *DataReaders* to match alongside Zero Copy v2 *DataReaders*, leave `TransportEncapsulationQosPolicy` at its default (empty sequence), which causes the middleware to create a CDR type plugin for every enabled transport.

### 4.9.7 Configuration

*Connext Micro* Zero Copy v2 includes some properties unique to its functionality. The following properties are always required:

- max_samples_per_notif
- user_intf[1]
- user_property[2]

The following properties are required if you are using the default implementation of the notification mechanism for Zero Copy v2. These are essentially a default set of user-defined properties; if you are using your own notification mechanism, you can set your own user-defined properties as needed.

- user_property.intf_addr
- user_property.thread_prop
  - user_property.thread_prop.stack_size
  - user_property.thread_prop.priority

[1] This property is only required if you choose to implement your own notification mechanism and not use the default implementation provided by RTI.

[2] Resolves to ZCOPY_NotifMechanismProperty when using the default notification mechanism.

  - user_property.thread_prop.options
- user_property.max_receive_ports
- user_property.max_routes

The following additional properties are only required if you are using your own notification mechanism for Zero Copy v2, not the default implementation.

- user_intf.create_instance
- user_intf.delete_instance
- user_intf.get_route_table
- user_intf.reserve_address
- user_intf.release_address
- user_intf.resolve_address
- user_intf.add_route
- user_intf.delete_route
- user_intf.bind
- user_intf.unbind
- user_intf.send
- user_intf.notify_recv_port
- user_intf.create_instance

### 4.9.8 Using multiple Zero Copy v2 transport instances

The platform-independent Zero Copy v2 transport supports multiple instances, provided that the user-defined, platform-specific implementation of the `notif` interface implements a way to uniquely identify each instance. In this case, each Zero Copy v2 transport instance should be registered with uniquely different names and properties.

When multiple instances of the Zero Copy v2 transport exist, individual *DataReaders* and *DataWriters* can be configured to use a specific instance of the Zero Copy v2 transport. This configuration is done in the entity's enabled_transports QoS configuration. For more information, see *Transport Registration*.

## 4.10 UDP Transport

This section describes the builtin *RTI Connext Micro* UDP transport and how to configure it.

The builtin UDP transport (UDP) is a fairly generic UDPv4 transport. *Connext Micro* supports the following functionality:

- Unicast
- Multicast
- Automatic detection of available network interfaces
- Manual configuration of network interfaces
- Allow/Deny lists to select which network interfaces can be used to receive data
- Simple NAT configuration
- Configuration of receive threads

---

**Note:** *Connext Micro* supports up to four network interfaces at once for each of the following:

- Unicast user-data
- Multicast user-data
- Unicast discovery data
- Multicast discovery data

---

### 4.10.1 Registering the UDP Transport

The builtin UDP transport is a *Connext Micro* component that must be registered. When a component is registered, the registration takes the properties and a listener as the 3rd and 4th parameters. In general, it is up to the caller to manage the memory for the properties and the listeners. There is no guarantee that a component makes a copy.

The following code snippet shows how to register the UDP transport:

```
struct UDP_InterfaceFactoryProperty *udp_property = NULL;

/* Allocate a property structure for the heap, it must be valid as long
 * as the component is registered
 */
udp_property = (struct UDP_InterfaceFactoryProperty *)
                    malloc(sizeof(struct UDP_InterfaceFactoryProperty));
if (udp_property != NULL)
{
    *udp_property = UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;

    /* Only allow network interface "eth0" to be used;
```

(continues on next page)

(continued from previous page)

```
     */
    REDA_StringSeq_set_maximum(&udp_property->allow_interface, 1);
    REDA_StringSeq_set_length(&udp_property->allow_interface, 1);

    *REDA_StringSeq_get_reference(&udp_property->allow_interface, 0) =
                                                    REDA_String_dup("eth0");

    /* Register the transport
     */
    if (!RT_Registry_register(registry, "_udp",
                          UDP_InterfaceFactory_get_interface(),
                         (struct RT_ComponentFactoryProperty*)udp_property,
                         NULL))
    {
        /* ERROR */
    }
}
else
{
    /* ERROR */
}
```

Note that the UDP transport can be registered with any name, but all transport properties and initial peers must refer to this name. If a transport is referred to and it does not exist, an error message is logged.

It is possible to register multiple UDP transports with a DomainParticipantFactory. It is also possible to use different UDP transports within the same *DomainParticipant* when multiple network interfaces are available (either physical or virtual).

When UDP transformations are enabled, this feature is always enabled and determined by the *allow_interface* and *deny_interface* lists. If any of the lists are non-empty the UDP transports will bind each receive socket to the specific interfaces.

When UDP transformations are not enabled, this feature is determined by the value of the *enable_interface_bind*. If this value is set to **RTI_TRUE** and the *allow_interface* and/or *deny_interface* properties are non-empty, the receive sockets are bound to specific interfaces.

### 4.10.2 Threading Model

The UDP transport creates one receive thread for each unique UDP receive address and port. Thus, by default, three UDP threads are created:

- A multicast receive thread for discovery data (assuming multicast is available and enabled)
- A unicast receive thread for discovery data
- A unicast receive thread for user data

Additional threads may be created depending on the transport configuration for a *DomainParticipant*, *DataReader*, and *DataWriter*. The UDP transport creates threads based on the following criteria:

- Each unique unicast port creates a new thread
- Each unique multicast address *and* port creates a new thread

For example, if a *DataReader* specifies its own multicast receive address, a new receive thread will be created.

### Configuring UDP Receive Threads

All threads in the UDP transport share the same thread settings. It is important to note that all the UDP properties must be set before the UDP transport is registered. *Connext Micro* preregisters the UDP transport with default settings when the DomainParticipantFactory is initialized. To change the UDP thread settings, use the following code.

```
struct UDP_InterfaceFactoryProperty *udp_property = NULL;
struct UDP_InterfaceFactoryProperty udp_property =
                              UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;

/* Allocate a property structure for the heap, it must be valid as long
 * as the component is registered
 */
udp_property = (struct UDP_InterfaceFactoryProperty *)
                     malloc(sizeof(struct UDP_InterfaceFactoryProperty));
*udp_property = UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;

/* Please refer to OSAPI_ThreadOptions for possible options */
udp_property->recv_thread.options = ...;

/* The stack-size is platform dependent, it is passed directly to the OS */
udp_property->recv_thread.stack_size = ....

/* The priority is platform dependent, it is passed directly to the OS */
udp_property->recv_thread.priority = ....

if (!RT_Registry_register(registry, "_udp",
                          UDP_InterfaceFactory_get_interface(),
                          (struct RT_ComponentFactoryProperty*)udp_property,
                          NULL))
{
    /* ERROR */
}
```

### 4.10.3 UDP Configuration

You can configure the UDP transport via the UDP_InterfaceFactoryProperty. The following fields are available:

#### allow_interface

The *allow_interface* string sequence determines which interfaces are allowed to be used for communication. Each string element is the name of a network interface, such as "en0" or "eth1".

If this sequence is empty, all interface names pass the allow test. The default value is empty. Thus, all interfaces are allowed.

#### deny_interface

The *deny_interface* string sequence determines which interfaces are not allowed to be used for communication. Each string element is the name of a network interface, such as "en0" or "eth1".

If this sequence is empty, the test is false. That is, the interface is allowed. Note that the deny list is checked *after* the allow list. Thus, if an interface appears in both, it is denied. The default value is empty, thus no interfaces are denied.

#### max_send_buffer_size

The *max_send_buffer_size* is the maximum size of the send socket buffer and it *must* be at least as big as the largest sample. Typically, this buffer should be a multiple of the maximum number of samples that can be sent at any given time. The default value is 256KB.

#### max_receive_buffer_size

The *max_receive_buffer_size* is the maximum size of the receive socket buffer and it *must* be at least as big as the largest sample. Typically, this buffer should be a multiple of the maximum number of samples that can be received at any given time. The default value is 256KB.

#### max_message_size

The *max_message_size* is the maximum size of the message which can be received, including any packet overhead. The default value is 8192 bytes.

#### multicast_ttl

The *multicast_ttl* is the Multicast Time-To-Live (TTL). This value is only used for multicast. It limits the number of hops a packet can pass through before it is dropped by a router. The default value is 1.

#### nat

*Connext Micro* supports firewalls with NAT. However, this feature has limited use and only supports translation between a private and public IP address. UDP ports are not translated. Furthermore, because *Connext Micro* does not support any hole punching technique or WAN server, this feature is only useful when the private and public address mapping is static and known in advance. For example, to test between an Android emulator and the host, the following configuration can be used:

```
UDP_NatEntrySeq_set_maximum(&udp_property->nat,2);
UDP_NatEntrySeq_set_length(&udp_property->nat,2);

/* Translate the local emulator  eth0 address 10.10.2.f:7410 to
 * 127.0.0.1:7410. This ensures that the address advertised by the
 * emulator to the host machine is the host's loopback interface, not
 * the emulator's host interface
 */
UDP_NatEntrySeq_get_reference(&udp_property->nat,0)->
                            local_address.kind = NETIO_ADDRESS_KIND_UDPv4;
UDP_NatEntrySeq_get_reference(&udp_property->nat,0)->
                            local_address.port = 7410;
UDP_NatEntrySeq_get_reference(&udp_property->nat,0)->
                            local_address.value.ipv4.address = 0x0a00020f;

UDP_NatEntrySeq_get_reference(&udp_property->nat,0)->
                            public_address.kind = NETIO_ADDRESS_KIND_UDPv4;
UDP_NatEntrySeq_get_reference(&udp_property->nat,0)->
                            public_address.port = 7410;
UDP_NatEntrySeq_get_reference(&udp_property->nat,0)->
                            public_address.value.ipv4.address = 0x7f000001;

/* Translate the local emulator  eth0 address 10.10.2.f:7411 to
 * 127.0.0.1:7411. This ensures that the address advertised by the
 * emulator to the host machine is the host's loopback interface
 */
UDP_NatEntrySeq_get_reference(&udp_property->nat,1)->
                            local_address.kind = NETIO_ADDRESS_KIND_UDPv4;
UDP_NatEntrySeq_get_reference(&udp_property->nat,1)->
                            local_address.port = 7411;
UDP_NatEntrySeq_get_reference(&udp_property->nat,1)->
                            local_address.value.ipv4.address = 0x0a00020f;

UDP_NatEntrySeq_get_reference(&udp_property->nat,1)->
                            public_address.kind = NETIO_ADDRESS_KIND_UDPv4;
UDP_NatEntrySeq_get_reference(&udp_property->nat,1)->
```

(continues on next page)

(continued from previous page)

```
                              public_address.port = 7411;
UDP_NatEntrySeq_get_reference(&udp_property->nat,1)->
                              public_address.value.ipv4.address = 0x7f000001;
```

#### if_table

The *if_table* provides a method to manually configure which interfaces are available for use; for example, when using IP stacks that do not support reading interface lists. The following example shows how to manually configure the interfaces.

```
/* The arguments to the UDP_InterfaceTable_add_entry functions are:
 * The if_table itself
 * The network address of the interface
 * The netmask of the interface
 * The name of the interface
 * Interface flags. Valid flags are:
 *   UDP_INTERFACE_INTERFACE_UP_FLAG        - The interface is UP
 *   UDP_INTERFACE_INTERFACE_MULTICAST_FLAG - The interface supports multicast
 */
if (!UDP_InterfaceTable_add_entry(&udp_property->if_table,
                                  0x7f000001,0xff000000,"loopback",
                                  UDP_INTERFACE_INTERFACE_UP_FLAG |
                                  UDP_INTERFACE_INTERFACE_MULTICAST_FLAG))

{
    /* Error */
}
```

#### multicast_interface

The *multicast_interface* may be used to select a particular network interface to be used to send multicast packets. The default value is any interface (that is, the OS selects the interface).

#### is_default_interface

The *is_default_interface* flag is used to indicate that this *Connext Micro* network transport shall be used if no other transport is found. The default value is **RTI_TRUE**.

#### disable_auto_interface_config

Normally, the UDP transport will try to read the interface list from the underlying OS (on platforms that support it). Setting *disable_auto_interface_config* to **RTI_TRUE** will prevent the UDP transport from reading the interface list.

#### multicast_loopback_disabled

The multicast_loopback_disabled field controls whether *Connext Micro* puts multicast packets onto the loopback interface.

#### recv_thread

The *recv_thread* field is used to configure all the receive threads. Please refer to *Threading Model* for details.

#### enable_interface_bind

When this is set to **TRUE** the UDP transport binds each receive port to a specific interface when the *allow_interface/deny_interface* lists are non-empty. This allows multiple UDP transports to be used by a single *DomainParticipant* at the expense of an increased number of threads. This property is ignored when transformations are enabled and the *allow_interface/deny_interface* lists are non-empty.

#### disable_multicast_bind

The disable_multicast_bind field controls whether *Connext Micro* will bind to a multicast address receive address (if set to 0) or bind to ANY multicast address (if set to 1).

#### disable_multicast_interface_select

The disable_multicast_interface_select field controls whether *Connext Micro* will use the *multicast_interface* (if specified), the *allow_interface/deny_interface* (if specified and *multicast_interface* is not specified) to select the interfaces used for sending to multicast addresses. If set to 1, *Connext Micro* will not select any interface.

#### source_rules

Rules for how to transform received UDP payloads based on the source address.

#### destination_rules

Rules for how to transform sent UDP payloads based on the destination address.

#### transform_udp_mode

Determines how regular UDP is supported when transformations are supported. When transformations are enabled the default value is **UDP_TRANSFORM_UDP_MODE_DISABLED**.

#### transform_locator_kind

The locator to use for locators that have transformations. When transformation rules have been enabled, they are announced as a vendor specific locator. This property overrides this value.

NOTE: Changing this value may prevent communication.

### 4.10.4 UDP Transformations

The UDP transform feature enables custom transformation of incoming and outgoing UDP payloads based on transformation rules between a pair of source and destination IP addresses. Some examples of transformations are encrypted data or logging.

This section explains how to implement and use transformations in an application and is organized as follows:

- *Overview*
- *Enabling UDP Transformation*
- *Creating a Transformation Library*
- *Creating Transformation Rules*
- *Interoperability*
- *Error Handling*
- *Example Code*
- *Examples*
- *OS Configuration*

### Overview

The UDP transformation feature enables custom transformation of incoming and outgoing UDP payloads. For the purpose of this section, a UDP payload is defined as a sequence of octets sent or received as a single UDP datagram excluding UDP headers – typically UDP port numbers – and trailers, such as the optional used checksum.

An outgoing payload is the UDP payload passed to the network stack. The transformation feature allows a custom transformation of this payload just before it is sent. The UDP transport receives payloads to send from an upstream layer. In *Connext Micro* this layer is typically RTPS, which creates payloads containing one or more RTPS messages. The transformation feature enables transformation of the entire RTPS payload before it is passed to the network stack.

The same RTPS payload may be sent to one or more locators. A locator identifies a destination address, such as an IPv4 address, a port, such as a UDP port, and a transport kind. The address and port are used by the UDP transport to reach a destination. However, only the destination address is used to determine which transformation to apply.

An incoming payload is the UDP payload received from the network stack. The transformation feature enables transformation of the UDP payload received from the network stack *before* it is passed to the upstream interface, typically RTPS. The UDP transport only receives payloads destined for one of its network interface addresses, but may receive UDP payloads destined for many different ports. The transformation does not take a port into account, only the source address. In *Connext Micro* the payload is typically a RTPS payload containing one or more RTPS messages.

UDP transformations are registered with *Connext Micro* and used by the UDP transport to determine how to transform payloads based on a source or destination address. Please refer to *Creating a Transformation Library* for details on how to implement transformations and *Creating Transformation Rules* for how to add rules.

Transformations are local resources. There is no exchange between different UDP transports regarding what a transformation does to a payload. This is considered a-priori knowledge and depends on the implementation of the transformation. Any negotiation of e.g. keys must be handled before the UDP transport is registered. Thus, if a sender and receiver do not apply consistent rules, they may not be able to communicate, or incorrect data may result. Note that while information is typically in the direction from a *DataWriter* to a *DataReader*, a reliable *DataReader* also send protocol data to a *DataWriter*. These messages are also transformed.

### Network Interface Selection

When a *DomainParticipant* is created, it first creates an instance of each transport configured in the DDS_DomainParticipantQos::transports QoS policy. Thus, each UDP transport registered with *Connext Micro* must have a unique name (up to 7 characters). Each registered transport can be configured to use all or some of the available interfaces using the *allow_interface* and *deny_interface* properties. The registered transports may now be used for either discovery data (specified in DomainParticipantQos::discovery), user_traffic (specified in DomainParticipantQos::user_traffic) or both. The *DomainParticipant* also queries the transport for which addresses it is capable of sending to.

When a participant creates multiple instances of the UDP transport, it is important that instances use non-overlapping networking interface resources.

### Data Reception

Which transport to use for discovery data is determined by the DomainParticipantQos::discovery QoS policy. For each transport listed, the *DomainParticipant* reserves a network address to listen to. This network address is sent as part of the discovery data and is used by other *DomainParticipants* as the address to send discovery data for this *DomainParticipant*. Because a UDP transformation only looks at source and destination addresses, if different transformations are needed for discovery and user-data, different UDP transport registrations must be used and hence different network interfaces.

### Data Transmission

Which address to send data to is based on the locators received as part of discovery and the peer list.

Received locators are analyzed and a transport locally registered with a *DomainParticipant* is selected based on the locator kind, address and mask. The first matching transport is selected. If a matching transport is not found, the locator is discarded.

NOTE: A transport is not a matching criteria at the same level as a QoS policy. If a discovered entity requests user data on a transport that doesn't exist, it is not unmatched.

The peer list, as specified by the application, is a list of locators to send participant discovery announcements to. If the transport to use is not specified, e.g. "udp1@192.168.1.1", but instead "192.168.1.1", then all transports that understand this address will send to it. Thus, in this case the latter is used, and two different UDP transports are registered; they will both send to the same address. However, one transport may send transformed data and the other may not depending on the destination address.

### Enabling UDP Transformation

The UDP transformation feature is not enabled by default. In order to use transformations, you must do the following:

1. Compile *Connext Micro* with the Cmake flag `-DRTIME_UDP_ENABLE_TRANSFORMS=TRUE`, and;
2. Compile your application with the `-DUDP_ENABLE_TRANSFORMS=1` C preprocessor flag.

---

**Note:** In order for UDP transformations to interoperate with a UDP transport without transformations enabled, you must specify the following UDP property:

```
udp_property->transform_udp_mode = UDP_TRANSFORM_UDP_MODE_ENABLED;
```

---

### Creating a Transformation Library

The transformation library is responsible for creating and performing transformations. Note that a library is a logical concept and does not refer to an actual library in an operating system. A library in this context is a collection of routines that together creates, manages, and performs transformations. How these routines are compiled and linked with an application using *Connext Micro* is out of scope of this section.

The transformation library must be registered with *Connext Micro*'s run-time and must implement the required interfaces. This ensures proper life-cycle management of transformation resources as well as clear guidelines regarding concurrency and memory management.

From *Connext Micro*'s run-time point of view, the transformation library must implement methods so that:

- A library can be initialized.
- A library can be instantiated.
- An instance of the library performs and manages transformations.

The first two tasks are handled by *Connext Micro*'s run-time factory interface which is common for all libraries managed by *Connext Micro*. The third task is handled by the transformation interface, which is specific to UDP transformations.

The following describes the relationship between the different interfaces:

- A library is initialized once when it is registered with *Connext Micro*.
- A library is finalized once when it is unregistered from *Connext Micro*.
- Multiple library instances can be created. If a library is used twice, for example registered with two different transports, two different library contexts are created using the factory interface. *Connext Micro* assumes that concurrent access to two different instances is allowed.
- Different instances of the library can be deleted independently. An instance is deleted using the factory interface.
- A library instance creates specific source or destination transformations. Each transformation is expected to transform a payload to exactly one destination or from one source.

The following relationship is true between the UDP transport and a UDP transformation library:

- Each registered UDP transport may make use of one or more UDP transformation libraries.
- A DDS *DomainParticipant* creates one instance of each registered UDP transport.
- Each instance of the UDP transport creates one instance of each enabled transformation library registered with the UDP transport.
- Each Transformation rule created by the UDP transport creates one send or one receive transformation.

### Creating Transformation Rules

Transformation rules decide how a payload should be transformed based on either a source or destination address. Before a UDP transport is registered, it must be configured with the transformation libraries to use, as well as which library to use for each source and destination address. For each UDP payload sent or received, an instance of the UDP transport searches for a matching source or destination rule to determine which transformation to apply.

The transformation rules are added to the UDP_InterfaceFactoryProperty before registration takes place.

If no transformation rules have been configured, all payloads are treated as regular UDP packets.

If no send rules have been asserted, the payload is sent as is. If all outgoing messages are to be transformed, a single entry is sufficient (address = 0, mask = 0).

If no receive rules have been asserted, it is passed upstream as is. If all incoming messages are to be transformed, a single entry is sufficient (address = 0, mask = 0).

If no matching rule is found, the packet is dropped and an error is logged.

NOTE: UDP_InterfaceFactoryProperty is immutable after the UDP transport has been registered.

### Interoperability

When the UDP transformations has enabled at least one transformation, it will only inter-operate with another UDP transport which also has at least one transformation.

UDP transformations does not interoperate with *RTI Connext Professional*.

### Error Handling

The transformation rules are applied on a local basis and correctness is based on configuration. It is not possible to detect that a peer participant is configured for different behavior and errors cannot be detected by the UDP transport itself. However, the transformation interface can return errors which are logged.

### Example Code

Example Header file MyUdpTransform.h:

```
#ifndef MyUdpTransform_h
#define MyUdpTransform_h

#include "rti_me_c.h"
#include "netio/netio_udp.h"
#include "netio/netio_interface.h"

struct MyUdpTransformFactoryProperty
```

(continues on next page)

(continued from previous page)

```
{
    struct RT_ComponentFactoryProperty _parent;
};

extern struct RT_ComponentFactoryI*
MyUdpTransformFactory_get_interface(void);

extern RTI_BOOL
MyUdpTransformFactory_register(RT_Registry_T *registry,
                               const char *const name,
                               struct MyUdpTransformFactoryProperty *property);

extern RTI_BOOL
MyUdpTransformFactory_unregister(RT_Registry_T *registry,
                 const char *const name,
                 struct MyUdpTransformFactoryProperty **);

#endif
```

Example Source file MyUdpTransform.c:

```
/*ce
 * \file
 * \defgroup UDPTransformExampleModule MyUdpTransform
 * \ingroup UserManuals_UDPTransform
 * \brief UDP Transform Example
 *
 * \details
 *
 * The UDP interface is implemented as a NETIO interface and NETIO interface
 * factory.
 */

 /*ce \addtogroup UDPTransformExampleModule
  * @{
  */
#include <stdio.h>

#include "MyUdpTransform.h"


/*ce
 * \brief The UDP Transformation factory class
 *
 * \details
 * All Transformation components must have a factory. A factory creates one
 * instance of the component as needed. In the case of UDP transformations,
 * \rtime creates one instance per UDP transport instance.
 */
struct MyUdpTransformFactory
{
```

(continues on next page)

(continued from previous page)

```
    /*ce
     * \brief Base-class. All \rtime Factories must inherit from RT_ComponentFactory.
     */
    struct RT_ComponentFactory _parent;

    /*ce
     * \brief A pointer to the properties of the factory.
     *
     * \details
     *
     * When a factory is registered with \rtime it can be registered with
     * properties specific to the component. However \rtime does not
     * make a copy ( that would require additional methods). Furthermore, it
     * may not be desirable to make a copy. Instead, this decision is
     * left to the implementer of the component. \rtime does not access
     * any custom properties.
     */
    struct MyUdpTransformFactoryProperty *property;
};

/*ce
 * \brief The custom UDP transformation class.
 *
 * \details
 * The MyUdpTransformFactory creates one instance of this class for each
 * UDP interface created. In this example one packet buffer (NETIO_Packet_T),
 * is allocated and a buffer to hold the transformed data (\ref buffer)
 *
 * Only one transformation can be done at a time and it is synchronous. Thus,
 * it is sufficient with one buffer to transform input and output per
 * instance of the MyUdpTransform.
 */
struct MyUdpTransform
{
    /*ce
     * \brief Base-class. All UDP transforms must inherit from UDP_Transform
     */
    struct UDP_Transform _parent;

    /*ce \brief A reference to its own factory, if properties must be accessed
     */
    struct MyUdpTransformFactory *factory;

    /*ce \brief NETIO_Packet to hold a transformed payload.
     *
     * \details
     *
     * \rtime uses a NETIO_Packet_T to abstract data payload and this is
     * what is being passed betweem the UDP transport and the transformation.
     * The transformation must convert a payload into a NETIO_Packet. This
     * is done with NETIO_Packet_initialize_from. This function saves all
```

(continues on next page)

(continued from previous page)

```
     * state except the payload buffer.
     */
    NETIO_Packet_T packet;

    /*ce \brief The payload to assign to NETIO_Packet_T
     *
     * \details
     *
     * A transformation cannot do in-place transformations because the input
     * buffer may be sent multiple times (for example due to reliability).
     * A transformation instance can only transform one buffer at a time
     * (send or receive). The buffer must be large enough to hold a transformed
     * payload. When the the transformation is created it receives a
     * \ref UDP_TransformProperty. This property has the max send and
     * receive buffers for transport and can be used to sise the buffer.
     * Please refer to \ref UDP_InterfaceFactoryProperty::max_send_message_size
     * and \ref UDP_InterfaceFactoryProperty::max_message_size.
     */
    char *buffer;

    /*ce \brief The maximum length of the buffer. NOTE: The buffer must
     * be 1 byte larger than the largest buffer.
     */
    RTI_SIZE_T max_buffer_length;
};

/*ce \brief Forward declaration of the interface implementation
 */
static struct UDP_TransformI MyUdpTransform_fv_Intf;

/*ce \brief Forward declaration of the interface factory implementation
 */
static struct RT_ComponentFactoryI MyUdpTransformFactory_fv_Intf;

/*ce \brief Method to create an instance of MyUdpTransform
 *
 * \param[in] factory The factory creating this instance
 * \param[in] property Generic UDP_Transform properties
 *
 * \return A pointer to MyUdpTransform on success, NULL on failure.
 */
RTI_PRIVATE struct MyUdpTransform*
MyUdpTransform_create(struct MyUdpTransformFactory *factory,
                      const struct UDP_TransformProperty *const property)
{
    struct MyUdpTransform *t;

    OSAPI_Heap_allocate_struct(&t, struct MyUdpTransform);
    if (t == NULL)
    {
        return NULL;
```

(continues on next page)

(continued from previous page)

```
    }

    /* All component instances must initialize the parent using this
     * call.
     */
    RT_Component_initialize(&t->_parent._parent,
                            &MyUdpTransform_fv_Intf._parent,
                            0,
                            (property ? &property->_parent : NULL),
                            NULL);

    t->factory = factory;

    /* Allocate a buffer that is the larger of the send and receive
     * size.
     */
    t->max_buffer_length = property->max_receive_message_size;
    if (property->max_send_message_size > t->max_buffer_length )
    {
        t->max_buffer_length = property->max_send_message_size;
    }

    /* Allocate 1 extra byte */
    OSAPI_Heap_allocate_buffer(&t->buffer,t->max_buffer_length+1,
                               OSAPI_ALIGNMENT_DEFAULT);

    if (t->buffer == NULL)
    {
        OSAPI_Heap_free_struct(t);
        t = NULL;
    }

    return t;
}

/*ce \brief Method to delete an instance of MyUdpTransform
 *
 * \param[in] t Transformation instance to delete
 */
RTI_PRIVATE void
MyUdpTransform_delete(struct MyUdpTransform *t)
{
    OSAPI_Heap_free_buffer(t->buffer);
    OSAPI_Heap_free_struct(t);
}

/*ce \brief Method to create a transformation for an destination address
 *
 * \details
 *
 * For each asserted destination rule a transform is created by the transformation
```

(continues on next page)

(continued from previous page)

```
 * instance. This method determines how a UDP payload is transformed before
 * it is sent to an address that matches destination & netmask.
 *
 * \param[in]  udptf       UDP Transform instance that creates the transformation
 * \param[out] context     Pointer to a transformation context
 * \param[in]  destination Destination address for the transformation
   \param[in]  netmask     The netmask to apply to this destination.
 * \param[in]  user_data   The user_data the rule was asserted with
 * \param[in]  property    UDP transform specific properties
 * \param[out] ec          User defined error code
 *
 * \return RTI_TRUE on success, RTI_FALSE on failure
 */
RTI_PRIVATE RTI_BOOL
MyUdpTransform_create_destination_transform(
                            UDP_Transform_T *const udptf,
                            void **const context,
                            const struct NETIO_Address *const destination,
                            const struct NETIO_Netmask *const netmask,
                            void *user_data,
                            const struct UDP_TransformProperty *const property,
                            RTI_INT32 *const ec)
{
    struct MyUdpTransform *self = (struct MyUdpTransform*)udptf;
    UNUSED_ARG(self);
    UNUSED_ARG(destination);
    UNUSED_ARG(user_data);
    UNUSED_ARG(property);
    UNUSED_ARG(ec);
    UNUSED_ARG(netmask);

    /* Save the user-data to determine which transform to apply later */
    *context = (void*)user_data;

    return RTI_TRUE;
}

/*ce \brief Method to delete a transformation for an destination address
 *
 *
 * \param[in]  udptf       UDP Transform instance that created the transformation
 * \param[out] context     Pointer to a transformation context
 * \param[in]  destination Destination address for the transformation
 * \param[in]  netmask     The netmask to apply to this destination.
 * \param[out] ec          User defined error code
 *
 * \return RTI_TRUE on success, RTI_FALSE on failure
 */
RTI_PRIVATE RTI_BOOL
MyUdpTransform_delete_destination_transform(UDP_Transform_T *const udptf,
                                void *context,
```

(continues on next page)

(continued from previous page)

```
                                const struct NETIO_Address *const destination,
                                const struct NETIO_Netmask *const netmask,
                                RTI_INT32 *const ec)
{
    UNUSED_ARG(udptf);
    UNUSED_ARG(context);
    UNUSED_ARG(destination);
    UNUSED_ARG(ec);
    UNUSED_ARG(netmask);

    return RTI_TRUE;
}

/*ce \brief Method to create a transformation for an source address
 *
 * \details
 *
 * For each asserted source rule a transform is created by the transformation
 * instance. This method determines how a UDP payload is transformed when
 * it is received from an address that matches source & netmask.
 *
 * \param[in]  udptf       UDP Transform instance that creates the transformation
 * \param[out] context     Pointer to a transformation context
 * \param[in]  source      Destination address for the transformation
   \param[in]  netmask     The netmask to apply to this destination.
 * \param[in]  user_data   The user_data the rule was asserted with
 * \param[in]  property    UDP transform specific properties
 * \param[out] ec          User defined error code
 *
 * \return RTI_TRUE on success, RTI_FALSE on failure
 */
RTI_PRIVATE RTI_BOOL
MyUdpTransform_create_source_transform(UDP_Transform_T *const udptf,
                            void **const context,
                            const struct NETIO_Address *const source,
                            const struct NETIO_Netmask *const netmask,
                            void *user_data,
                            const struct UDP_TransformProperty *const property,
                            RTI_INT32 *const ec)
{
    struct MyUdpTransform *self = (struct MyUdpTransform*)udptf;
    UNUSED_ARG(self);
    UNUSED_ARG(source);
    UNUSED_ARG(user_data);
    UNUSED_ARG(property);
    UNUSED_ARG(ec);
    UNUSED_ARG(netmask);

    *context = (void*)user_data;

    return RTI_TRUE;
```

(continues on next page)

(continued from previous page)

```
}

/*ce \brief Method to delete a transformation for an source address
 *
 *
 * \param[in]  udptf       UDP Transform instance that created the transformation
 * \param[out] context     Pointer to a transformation context
 * \param[in]  source      Source address for the transformation
 * \param[in]  netmask     The netmask to apply to this destination.
 * \param[out] ec          User defined error code
 *
 * \return RTI_TRUE on success, RTI_FALSE on failure
 */
RTI_PRIVATE RTI_BOOL
MyUdpTransform_delete_source_transform(UDP_Transform_T *const udptf,
                                       void *context,
                                       const struct NETIO_Address *const source,
                                       const struct NETIO_Netmask *const netmask,
                                       RTI_INT32 *const ec)
{
    UNUSED_ARG(udptf);
    UNUSED_ARG(context);
    UNUSED_ARG(source);
    UNUSED_ARG(ec);
    UNUSED_ARG(netmask);

    return RTI_TRUE;
}

/*ce \brief Method to transform data based on a source address
 *
 * \param[in]  udptf       UDP_Transform_T that performs the transformation
 * \param[in]  context     Reference to context created by \ref MyUdpTransform_create_
↪source_transform
 * \param[in]  source      Source address for the transformation
 * \param[in]  in_packet   The NETIO packet to transform
 * \param[out] out_packet  The transformed NETIO packet
 * \param[out] ec          User defined error code
 *
 * \return RTI_TRUE on success, RTI_FALSE on failure
 */
RTI_PRIVATE RTI_BOOL
MyUdpTransform_transform_source(UDP_Transform_T *const udptf,
                                void *context,
                                const struct NETIO_Address *const source,
                                const NETIO_Packet_T *const in_packet,
                                NETIO_Packet_T **out_packet,
                                RTI_INT32 *const ec)
{
    struct MyUdpTransform *self = (struct MyUdpTransform*)udptf;
    char *buf_ptr,*buf_end;
```

(continues on next page)

(continued from previous page)

```
    char *from_buf_ptr,*from_buf_end;
    UNUSED_ARG(context);
    UNUSED_ARG(source);

    *ec = 0;

    /* Assigned the transform buffer to the outgoing packet
     * saving state from the incoming packet. In this case the
     * outgoing length is the same as the incoming. How to buffer
     * is filled in is of no interest to \rtime. All it cares about is
     * where it starts and where it ends.
     */
    if (!NETIO_Packet_initialize_from(
                                &self->packet,in_packet,
                                self->buffer,self->max_buffer_length,
                                0,NETIO_Packet_get_payload_length(in_packet)))
    {
        return RTI_FALSE;
    }

    *out_packet = &self->packet;

    buf_ptr = NETIO_Packet_get_head(&self->packet);
    buf_end = NETIO_Packet_get_tail(&self->packet);
    from_buf_ptr = NETIO_Packet_get_head(in_packet);
    from_buf_end = NETIO_Packet_get_tail(in_packet);

    /* Perform a transformation based on the user-data */
    while (from_buf_ptr < from_buf_end)
    {
        if (context == (void*)1)
        {
            *buf_ptr = ~(*from_buf_ptr);
        }
        else if (context == (void*)2)
        {
            *buf_ptr = (*from_buf_ptr)+1;
        }

        ++buf_ptr;
        ++from_buf_ptr;
    }

    return RTI_TRUE;
}

/*ce \brief Method to transform data based on a destination address
 *
 * \param[in]  udptf       UDP_Transform_T that performs the transformation
 * \param[in]  context     Reference to context created by \ref MyUdpTransform_create_
↪destination_transform
```

(continues on next page)

(continued from previous page)

```
 * \param[in]  destination Source address for the transformation
 * \param[in]  in_packet   The NETIO packet to transform
 * \param[out] packet_out  The transformed NETIO packet
 * \param[out] ec          User defined error code
 *
 * \return RTI_TRUE on success, RTI_FALSE on failure
 */
RTI_PRIVATE RTI_BOOL
MyUdpTransform_transform_destination(UDP_Transform_T *const udptf,
                                     void *context,
                                     const struct NETIO_Address *const destination,
                                     const NETIO_Packet_T *const in_packet,
                                     NETIO_Packet_T **packet_out,
                                     RTI_INT32 *const ec)
{
    struct MyUdpTransform *self = (struct MyUdpTransform*)udptf;
    char *buf_ptr,*buf_end;
    char *from_buf_ptr,*from_buf_end;
    NETIO_Packet_T **out_packet = packet_out;
    UNUSED_ARG(context);
    UNUSED_ARG(destination);

    *ec = 0;

    if (!NETIO_Packet_initialize_from(
                                &self->packet,in_packet,
                                self->buffer,8192,
                                0,NETIO_Packet_get_payload_length(in_packet)))
    {
        return RTI_FALSE;
    }

    *out_packet = &self->packet;

    buf_ptr = NETIO_Packet_get_head(&self->packet);
    buf_end = NETIO_Packet_get_tail(&self->packet);
    from_buf_ptr = NETIO_Packet_get_head(in_packet);
    from_buf_end = NETIO_Packet_get_tail(in_packet);

    while (from_buf_ptr < from_buf_end)
    {
        if (context == (void*)1)
        {
            *buf_ptr = ~(*from_buf_ptr);
        }
        else if (context == (void*)2)
        {
            *buf_ptr = (*from_buf_ptr)-1;
        }

        ++buf_ptr;
```

(continues on next page)

(continued from previous page)

```
        ++from_buf_ptr;
    }

    return RTI_TRUE;
}

/*ce \brief Definition of the transformation interface
 */
RTI_PRIVATE struct UDP_TransformI MyUdpTransform_fv_Intf =
{
    RT_COMPONENTI_BASE,
    MyUdpTransform_create_destination_transform,
    MyUdpTransform_create_source_transform,
    MyUdpTransform_transform_source,
    MyUdpTransform_transform_destination,
    MyUdpTransform_delete_destination_transform,
    MyUdpTransform_delete_source_transform
};

/*ce \brief Method called by \rtime to create an instance of transformation
 */
MUST_CHECK_RETURN RTI_PRIVATE RT_Component_T*
MyUdpTransformFactory_create_component(struct RT_ComponentFactory *factory,
                        struct RT_ComponentProperty *property,
                        struct RT_ComponentListener *listener)
{
    struct MyUdpTransform *t;
    UNUSED_ARG(listener);

    t = MyUdpTransform_create(
                (struct MyUdpTransformFactory*)factory,
                (struct UDP_TransformProperty*)property);

    return &t->_parent._parent;
}

/*ce \brief Method called by \rtime to delete an instance of transformation
 */
RTI_PRIVATE void
MyUdpTransformFactory_delete_component(
                                        struct RT_ComponentFactory *factory,
                                        RT_Component_T *component)
{
    UNUSED_ARG(factory);

    MyUdpTransform_delete((struct MyUdpTransform*)component);
}

/*ce \brief Method called by \rtime when a factory is registered
 */
MUST_CHECK_RETURN RTI_PRIVATE struct RT_ComponentFactory*
```

(continues on next page)

(continued from previous page)

```
MyUdpTransformFactory_initialize(struct RT_ComponentFactoryProperty* property,
                                 struct RT_ComponentFactoryListener *listener)
{
    struct MyUdpTransformFactory *fac;
    UNUSED_ARG(property);
    UNUSED_ARG(listener);

    OSAPI_Heap_allocate_struct(&fac,struct MyUdpTransformFactory);

    fac->_parent._factory = &fac->_parent;
    fac->_parent.intf = &MyUdpTransformFactory_fv_Intf;
    fac->property = (struct MyUdpTransformFactoryProperty*)property;

    return &fac->_parent;
}

/*ce \brief Method called by \rtime when a factory is unregistered
 */
RTI_PRIVATE void
MyUdpTransformFactory_finalize(struct RT_ComponentFactory *factory,
                        struct RT_ComponentFactoryProperty **property,
                        struct RT_ComponentFactoryListener **listener)
{
    struct MyUdpTransformFactory *fac =
            (struct MyUdpTransformFactory*)factory;

    UNUSED_ARG(property);
    UNUSED_ARG(listener);

    if (listener != NULL)
    {
        *listener = NULL;
    }

    if (property != NULL)
    {
        *property = (struct RT_ComponentFactoryProperty*)fac->property;
    }

    OSAPI_Heap_free_struct(factory);

    return;
}

/*ce \brief Definition of the factory interface
 */
RTI_PRIVATE struct RT_ComponentFactoryI MyUdpTransformFactory_fv_Intf =
{
    UDP_INTERFACE_INTERFACE_ID,
    MyUdpTransformFactory_initialize,
    MyUdpTransformFactory_finalize,
```

(continues on next page)

(continued from previous page)

```
    MyUdpTransformFactory_create_component,
    MyUdpTransformFactory_delete_component,
    NULL,
    NULL
};

struct RT_ComponentFactoryI*
MyUdpTransformFactory_get_interface(void)
{
    return &MyUdpTransformFactory_fv_Intf;
}

/*ce \brief Method to register this transformation in a registry
 */
RTI_BOOL
MyUdpTransformFactory_register(RT_Registry_T *registry,
                               const char *const name,
                               struct MyUdpTransformFactoryProperty *property)
{
    return RT_Registry_register(registry, name,
                        MyUdpTransformFactory_get_interface(),
                        &property->_parent, NULL);
}

/*ce \brief Method to unregister this transformation from a registry
 */
RTI_BOOL
MyUdpTransformFactory_unregister(RT_Registry_T *registry,
            const char *const name,
            struct MyUdpTransformFactoryProperty **property)
{
    return RT_Registry_unregister(registry, name,
                                  (struct RT_ComponentFactoryProperty**)property,
                                  NULL);
}

/*! @} */
```

Example configuration of rules:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "common.h"

void
MyAppApplication_help(char *appname)
{
    printf("%s [options]\n", appname);
```

(continues on next page)

(continued from previous page)

```
    printf("options:\n");
    printf("-h                  - This text\n");
    printf("-domain <id>        - DomainId (default: 0)\n");
    printf("-udp_intf <intf>    - udp interface (no default)\n");
    printf("-peer <address>     - peer address (no default)\n");
    printf("-count <count>      - count (default -1)\n");
    printf("-sleep <ms>         - sleep between sends (default 1s)\n");
    printf("\n");
}

struct MyAppApplication*
MyAppApplication_create(const char *local_participant_name,
                        const char *remote_participant_name,
                        DDS_Long domain_id, char *udp_intf, char *peer,
                        DDS_Long sleep_time, DDS_Long count)
{
    DDS_ReturnCode_t retcode;
    DDS_DomainParticipantFactory *factory = NULL;
    struct DDS_DomainParticipantFactoryQos dpf_qos =
        DDS_DomainParticipantFactoryQos_INITIALIZER;
    struct DDS_DomainParticipantQos dp_qos =
        DDS_DomainParticipantQos_INITIALIZER;
    DDS_Boolean success = DDS_BOOLEAN_FALSE;
    struct MyAppApplication *application = NULL;
    RT_Registry_T *registry = NULL;
    struct UDP_InterfaceFactoryProperty *udp_property = NULL;
    struct DPDE_DiscoveryPluginProperty discovery_plugin_properties =
        DPDE_DiscoveryPluginProperty_INITIALIZER;
    UNUSED_ARG(local_participant_name);
    UNUSED_ARG(remote_participant_name);

    /* Uncomment to increase verbosity level:
       OSAPILog_set_verbosity(OSAPI_LOG_VERBOSITY_WARNING);
     */
    application = (struct MyAppApplication *)malloc(sizeof(struct MyAppApplication));

    if (application == NULL)
    {
        printf("failed to allocate application\n");
        goto done;
    }

    application->sleep_time = sleep_time;
    application->count = count;

    factory = DDS_DomainParticipantFactory_get_instance();

    if (DDS_DomainParticipantFactory_get_qos(factory,&dpf_qos) != DDS_RETCODE_OK)
    {
        printf("failed to get number of components\n");
        goto done;
```

(continues on next page)

(continued from previous page)

```
    }

    dpf_qos.resource_limits.max_components = 128;

    if (DDS_DomainParticipantFactory_set_qos(factory,&dpf_qos) != DDS_RETCODE_OK)
    {
        printf("failed to increase number of components\n");
        goto done;
    }

    registry = DDS_DomainParticipantFactory_get_registry(
                               DDS_DomainParticipantFactory_get_instance());

    if (!RT_Registry_register(registry, DDSHST_WRITER_DEFAULT_HISTORY_NAME,
                              WHSM_HistoryFactory_get_interface(), NULL, NULL))
    {
        printf("failed to register wh\n");
        goto done;
    }

    if (!RT_Registry_register(registry, DDSHST_READER_DEFAULT_HISTORY_NAME,
                              RHSM_HistoryFactory_get_interface(), NULL, NULL))
    {
        printf("failed to register rh\n");
        goto done;
    }

    if (!MyUdpTransformFactory_register(registry,"T0",NULL))
    {
        printf("failed to register T0\n");
        goto done;
    }

    if (!MyUdpTransformFactory_register(registry,"T1",NULL))
    {
        printf("failed to register T0\n");
        goto done;
    }

    /* Configure UDP transport's allowed interfaces */
    if (!RT_Registry_unregister(registry, NETIO_DEFAULT_UDP_NAME, NULL, NULL))
    {
        printf("failed to unregister udp\n");
        goto done;
    }

    udp_property = (struct UDP_InterfaceFactoryProperty *)
                            malloc(sizeof(struct UDP_InterfaceFactoryProperty));
    if (udp_property == NULL)
    {
        printf("failed to allocate udp properties\n");
```

(continues on next page)

(continued from previous page)

```
    goto done;
}
*udp_property = UDP_INTERFACE_FACTORY_PROPERTY_DEFAULT;

/* For additional allowed interface(s), increase maximum and length, and
   set interface below:
*/
udp_property->max_send_message_size = 16384;
udp_property->max_message_size = 32768;

if (udp_intf != NULL)
{
    REDA_StringSeq_set_maximum(&udp_property->allow_interface,1);
    REDA_StringSeq_set_length(&udp_property->allow_interface,1);
    *REDA_StringSeq_get_reference(&udp_property->allow_interface,0) =
            DDS_String_dup(udp_intf);
}


/* A rule that says: For payloads received from 192.168.10.* (netmask is
 * 0xffffff00), apply transformation T0.
 */
if (!UDP_TransformRules_assert_source_rule(
        &udp_property->source_rules,
        0xc0a80ae8,0xffffff00,"T0",(void*)2))
{
    printf("Failed to assert source rule\n");
    goto done;
}

/* A rule that says: For payloads sent to 192.168.10.* (netmask is
 * 0xffffff00), apply transformation T0.
 */
if (!UDP_TransformRules_assert_destination_rule(
        &udp_property->destination_rules,
        0xc0a80ae8,0xffffff00,"T0",(void*)2))
{
    printf("Failed to assert source rule\n");
    goto done;
}

/* A rule that says: For payloads received from 192.168.20.* (netmask is
 * 0xffffff00), apply transformation T1.
 */
if (!UDP_TransformRules_assert_source_rule(
        &udp_property->source_rules,
        0xc0a81465,0xffffff00,"T1",(void*)1))
{
    printf("Failed to assert source rule\n");
    goto done;
}
```

(continues on next page)

(continued from previous page)

```
    /* A rule that says: For payloads received from 192.168.20.* (netmask is
     * 0xffffff00), apply transformation T1.
     */
    if (!UDP_TransformRules_assert_destination_rule(
            &udp_property->destination_rules,
            0xc0a81465,0xffffff00,"T1",(void*)1))
    {
        printf("Failed to assert source rule\n");
        goto done;
    }

    if (!RT_Registry_register(registry, NETIO_DEFAULT_UDP_NAME,
                          UDP_InterfaceFactory_get_interface(),
                         (struct RT_ComponentFactoryProperty*)udp_property, NULL))
    {
       printf("failed to register udp\n");
       goto done;
    }

    DDS_DomainParticipantFactory_get_qos(factory, &dpf_qos);
    dpf_qos.entity_factory.autoenable_created_entities = DDS_BOOLEAN_FALSE;
    DDS_DomainParticipantFactory_set_qos(factory, &dpf_qos);

    if (peer == NULL)
    {
        peer = "127.0.0.1"; /* default to loopback */
    }

    if (!RT_Registry_register(registry,
                              "dpde",
                              DPDE_DiscoveryFactory_get_interface(),
                              &discovery_plugin_properties._parent,
                              NULL))
    {
        printf("failed to register dpde\n");
        goto done;
    }

    if (!RT_ComponentFactoryId_set_name(&dp_qos.discovery.discovery.name,"dpde"))
    {
        printf("failed to set discovery plugin name\n");
        goto done;
    }

    DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers,1);
    DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers,1);
    *DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers,0) = DDS_String_
↪dup(peer);

    DDS_StringSeq_set_maximum(&dp_qos.discovery.enabled_transports,1);
```

(continues on next page)

(continued from previous page)

```
    DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports,1);

    /* Use network interface 192.168.10.232 for discovery. T0 is used for
     * discovery
     */
    *DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports,0) = DDS_String_
↪dup("_udp://192.168.10.232");

    DDS_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports,1);
    DDS_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports,1);

    /* Use network interface 192.168.20.101 for user-data. T1 is used for
     * this interface.
     */
    *DDS_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports,0) = DDS_String_
↪dup("_udp://192.168.20.101");

    /* if there are more remote or local endpoints, you need to increase these limits */
    dp_qos.resource_limits.max_destination_ports = 32;
    dp_qos.resource_limits.max_receive_ports = 32;
    dp_qos.resource_limits.local_topic_allocation = 1;
    dp_qos.resource_limits.local_type_allocation = 1;
    dp_qos.resource_limits.local_reader_allocation = 1;
    dp_qos.resource_limits.local_writer_allocation = 1;
    dp_qos.resource_limits.remote_participant_allocation = 8;
    dp_qos.resource_limits.remote_reader_allocation = 8;
    dp_qos.resource_limits.remote_writer_allocation = 8;

    application->participant =
        DDS_DomainParticipantFactory_create_participant(factory, domain_id,
                                                        &dp_qos, NULL,
                                                        DDS_STATUS_MASK_NONE);

    if (application->participant == NULL)
    {
        printf("failed to create participant\n");
        goto done;
    }

    sprintf(application->type_name, "HelloWorld");
    retcode = DDS_DomainParticipant_register_type(application->participant,
                                                  application->type_name,
                                                  HelloWorldTypePlugin_get());
    if (retcode != DDS_RETCODE_OK)
    {
        printf("failed to register type: %s\n", "test_type");
        goto done;
    }

    sprintf(application->topic_name, "HelloWorld");
    application->topic =
```

(continues on next page)

(continued from previous page)

```
        DDS_DomainParticipant_create_topic(application->participant,
                                           application->topic_name,
                                           application->type_name,
                                           &DDS_TOPIC_QOS_DEFAULT, NULL,
                                           DDS_STATUS_MASK_NONE);

    if (application->topic == NULL)
    {
        printf("topic == NULL\n");
        goto done;
    }

    success = DDS_BOOLEAN_TRUE;

  done:

    if (!success)
    {
        if (udp_property != NULL)
        {
            free(udp_property);
        }
        free(application);
        application = NULL;
    }

    return application;
}

DDS_ReturnCode_t
MyAppApplication_enable(struct MyAppApplication * application)
{
    DDS_Entity *entity;
    DDS_ReturnCode_t retcode;

    entity = DDS_DomainParticipant_as_entity(application->participant);

    retcode = DDS_Entity_enable(entity);
    if (retcode != DDS_RETCODE_OK)
    {
        printf("failed to enable entity\n");
    }

    return retcode;
}

void
MyAppApplication_delete(struct MyAppApplication *application)
{
    DDS_ReturnCode_t retcode;
    RT_Registry_T *registry = NULL;
```

(continues on next page)

(continued from previous page)

```
    retcode = DDS_DomainParticipant_delete_contained_entities(application->participant);
    if (retcode != DDS_RETCODE_OK)
    {
        printf("failed to delete conteined entities (retcode=%d)\n",retcode);
    }

    if (DDS_DomainParticipant_unregister_type(application->participant,
                    application->type_name) != HelloWorldTypePlugin_get())
    {
        printf("failed to unregister type: %s\n", application->type_name);
        return;
    }

    retcode = DDS_DomainParticipantFactory_delete_participant(
                                DDS_DomainParticipantFactory_get_instance(),
                                application->participant);

    if (retcode != DDS_RETCODE_OK)
    {
        printf("failed to delete participant: %d\n", retcode);
        return;
    }

    registry = DDS_DomainParticipantFactory_get_registry(
                                DDS_DomainParticipantFactory_get_instance());

    if (!RT_Registry_unregister(registry, "dpde", NULL, NULL))
    {
        printf("failed to unregister dpde\n");
        return;
    }
    if (!RT_Registry_unregister(registry, DDSHST_READER_DEFAULT_HISTORY_NAME, NULL,␣
↪NULL))
    {
        printf("failed to unregister rh\n");
        return;
    }
    if (!RT_Registry_unregister(registry, DDSHST_WRITER_DEFAULT_HISTORY_NAME, NULL,␣
↪NULL))
    {
        printf("failed to unregister wh\n");
        return;
    }

    free(application);

    DDS_DomainParticipantFactory_finalize_instance();
}
```

### Examples

The following examples illustrate how this feature can be used in a system with a mixture of different types of UDP transport configurations.

For the purpose of the examples, the following terminology is used:

- Plain communication – No transformations have been applied.
- Transformed User Data – Only the user-data is transformed, discovery is plain.
- Transformed Discovery – Only the discovery data is transformed, user-data is plain.
- Transformed Data – Both discovery and user-data are transformed. Unless stated otherwise the transformations are different.

A transformation Tn is a transformation such that an outgoing payload transformed with Tn can be transformed back to its original state by applying Tn to the incoming data.

A network interface can be either physical or virtual.

### Plain Communication Between 2 Nodes

In this system two Nodes, A and B, are communicating with plain communication. Node A has one interface, a0, and Node B has one interface, b0.

Node A:

- Register the UDP transport Ua with allow_interface = a0.
- DomainParticipantQos.transports.enabled_transports = “Ua”
- DomainParticipantQos.discovery.enabled_transports = ”Ua://”
- DomainParticipantQos.user_data.enabled_transports = ”Ua://”

Node B:

- Register the UDP transport Ub with allow_interface = b0.
- DomainParticipantQos.transports.enabled_transports = “Ub”
- DomainParticipantQos.discovery.enabled_transports = ”Ub://”
- DomainParticipantQos.user_data.enabled_transports = ”Ub://”

#### Transformed User Data Between 2 Nodes

In this system two Nodes, A and B, are communicating with transformed user data. Node A has two interfaces, a0 and a1, and Node B has two interfaces, b0 and b1. Since each node has only one peer, a single transformation is sufficient.

Node A:

- Add a destination transformation T0 to Ua0, indicating that all sent data is transformed with T0.
- Add a source transformation T1 to Ua0, indicating that all received data is transformed with T1.
- Register the UDP transport Ua0 with allow_interface = a0.
- Register the UDP transport Ua1 with allow_interface = a1.
- No transformations are registered with Ua1.
- DomainParticipantQos.transports.enabled_transports = “Ua0”,”Ua1”
- DomainParticipantQos.discovery.enabled_transports = ”Ua1://”
- DomainParticipantQos.user_traffic.enabled_transports = ”Ua0://”

Node B:

- Add a destination transformation T1 to Ub0, indicating that all sent data is transformed with T1.
- Add a source transformation T0 to Ub0, indicating that all received data is transformed with T0.
- Register the UDP transport Ub0 with allow_interface = b0.
- Register the UDP transport Ub1 with allow_interface = b1.
- No transformations are registered with Ub1.
- DomainParticipantQos.transports.enabled_transports = “Ub0”,”Ub1”
- DomainParticipantQos.discovery.enabled_transports = ”Ub1://”
- DomainParticipantQos.user_traffic.enabled_transports = ”Ub0://”

Ua0 and Ub0 perform transformations and are used for user-data. Ua1 and Ub1 are used for discovery and no transformations takes place.

#### Transformed Discovery Data Between 2 Nodes

In this system two Nodes, A and B, are communicating with transformed user data. Node A has two interfaces, a0 and a1, and Node B has two interfaces, b0 and b1. Since each node has only one peer, a single transformation is sufficient.

Node A:

- Add a destination transformation T0 to Ua0, indicating that all sent data is transformed with T0.
- Add a source transformation T1 to Ua0, indicating that all received data is transformed with T1.
- Register the UDP transport Ua0 with allow_interface = a0.
- Register the UDP transport Ua1 with allow_interface = a1.
- No transformations are registered with Ua1.
- DomainParticipantQos.transports.enabled_transports = “Ua0”,”Ua1”
- DomainParticipantQos.discovery.enabled_transports = ”Ua0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ua1://”

Node B:

- Add a destination transformation T1 to Ub0, indicating that all sent data is transformed with T1.
- Add a source transformation T0 to Ub0, indicating that all received data is transformed with T0.
- Register the UDP transport Ub0 with allow_interface = b0.
- Register the UDP transport Ub1 with allow_interface = b1.
- No transformations are registered with Ub1.
- DomainParticipantQos.transports.enabled_transports = “Ub0”,”Ub1”
- DomainParticipantQos.discovery.enabled_transports = ”Ub0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ub1://”

Ua0 and Ub0 perform transformations and are used for discovery. Ua1 and Ub1 are used for user-data and no transformation takes place.

#### Transformed Data Between 2 Nodes (same transformation)

In this system two Nodes, A and B, are communicating with transformed data using the same transformation for user and discovery data. Node A has one interface, a0, and Node B has one interface, b0.

Node A:

- Add a destination transformation T0 to Ua0, indicating that all sent data is transformed with T0.
- Add a source transformation T1 to Ua0, indicating that all received data is transformed with T1.
- Register the UDP transport Ua0 with allow_interface = a0.
- DomainParticipantQos.transports.enabled_transports = “Ua0”
- DomainParticipantQos.discovery.enabled_transports = ”Ua0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ua0://”

Node B:

- Add a destination transformation T1 to Ub0, indicating that all sent data is transformed with T1.
- Add a source transformation T0 to Ub0, indicating that all received data is transformed with T0.
- Register the UDP transport Ub0 with allow_interface = b0.
- DomainParticipantQos.transports.enabled_transports = “Ub0”
- DomainParticipantQos.discovery.enabled_transports = ”Ub0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ub0://”

Ua0 and Ub0 performs transformations and are used for discovery and for user-data.

#### Transformed Data Between 2 Nodes (different transformations)

In this system two Nodes, A and B, are communicating with transformed data using different transformations for user and discovery data. Node A has two interfaces, a0 and a1, and Node B has two interfaces, b0 and b1.

Node A:

- Add a destination transformation T0 to Ua0, indicating that all sent data is transformed with T0.
- Add a source transformation T1 to Ua0, indicating that all received data is transformed with T1.
- Add a destination transformation T2 to Ua1, indicating that all sent data is transformed with T2.

- Add a source transformation T3 to Ua1, indicating that all received data is transformed with T3.
- Register the UDP transport Ua0 with allow_interface = a0.
- Register the UDP transport Ua1 with allow_interface = a1.
- DomainParticipantQos.transports.enabled_transports = “Ua0”,”Ua1”
- DomainParticipantQos.discovery.enabled_transports = ”Ua0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ua1://”

Node B:

- Add a destination transformation T1 to Ub0, indicating that all sent data is transformed with T1.
- Add a source transformation T0 to Ub0, indicating that all received data is transformed with T0.
- Add a destination transformation T3 to Ub1, indicating that all sent data is transformed with T3.
- Add a source transformation T2 to Ub1, indicating that all received data is transformed with T2.
- Register the UDP transport Ub0 with allow_interface = b0.
- Register the UDP transport Ub1 with allow_interface = b1.
- DomainParticipantQos.transports.enabled_transports = “Ub0”,”Ub1”
- DomainParticipantQos.discovery.enabled_transports = ”Ub0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ub1://”

Ua0 and Ub0 perform transformations and are used for discovery. Ua1 and Ub1 perform transformations and are used for user-data.

### OS Configuration

In systems with several network interfaces, *Connext Micro* cannot ensure which network interface should be used to send a packet. Depending on the UDP transformations configured, this might be a problem.

To illustrate this problem, let's assume a system with two nodes, A and B. Node A has two network interfaces, a0 and a1, and Node B has two network interfaces, b0 and b1. In this system, Node A is communicating with Node B using a transformation for discovery and a different transformation for user data.

Node A:

- Add a destination transformation T0 to Ua0, indicating that sent data to b0 is transformed with T0.

- Add a source transformation T1 to Ua0, indicating that received data from b0 is transformed with T1.
- Add a destination transformation T2 to Ua1, indicating that sent data to b1 is transformed with T2.
- Add a source transformation T3 to Ua1, indicating that received data from b1 is transformed with T3.
- Register the UDP transport Ua0 with allow_interface = a0.
- Register the UDP transport Ua1 with allow_interface = a1.
- DomainParticipantQos.transports.enabled_transports = “Ua0”,”Ua1”
- DomainParticipantQos.discovery.enabled_transports = ”Ua0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ua1://”

Node B:

- Add a destination transformation T1 to Ub0, indicating that sent data to a0 is transformed with T1.
- Add a source transformation T0 to Ub0, indicating that received data from a0 transformed with T0.
- Add a destination transformation T3 to Ub1, indicating that sent data to a1 is transformed with T3.
- Add a source transformation T2 to Ub1, indicating that received data from a1 transformed with T2.
- Register the UDP transport Ub0 with allow_interface = b0.
- Register the UDP transport Ub1 with allow_interface = b1.
- DomainParticipantQos.transports.enabled_transports = “Ub0”,”Ub1”
- DomainParticipantQos.discovery.enabled_transports = ”Ub0://”
- DomainParticipantQos.user_data.enabled_transports = ”Ub1://”

Node A sends a discovery packet to Node B to interface b0. This packet will be transformed using T0 as specified by Node A’s configuration. When this packet is received in Node B, it will be transformed using either T0 or T2 depending on the source address. Node’s A OS will use a0 or a1 to send this packet but *Connext Micro* cannot ensure which one will be used. In case the OS sends the packet using a1, the wrong transformation will be applied in Node B.

Some systems have the possibility to configure the source address that should be used when a packet is sent. In POSIX systems, the command `ip route add <string> dev <interface>` can be used.

By typing the command `ip route add < b0 ip >/32 dev a0` in Node A, the OS will send all packets to Node B’s b0 IP address using interface a0. This would ensure that the correct transformation is applied in Node B. The same should be done to ensure that user data is sent with the right address `ip route add < b1 ip >/32 dev a1`. Of course, similar configuration is needed in Node B.

## 4.11 NETIO Datagram Transport

This section describes the built-in *Connext Micro* Datagram transport and how to configure it.

The built-in Datagram transport (DGRAM) is a generic transport plugin service.

DGRAM is part of the *RTI Connext Micro* core library that is compiled for a specific CPU architecture with a specific compiler. However, the DGRAM transport does not include integration with any particular network stack. Instead, the DGRAM transport provides a simplified interface which can integrate with a variety of different networking technologies.

The DGRAM plugin supports transmission and reception of RTPS messages over a connectionless network link. Note that while the DGRAM transport itself has no knowledge of the underlying network stack, the DGRAM API does not include API related to establishing connections, such as TCP.

### 4.11.1 Registering a Datagram Interface

DGRAM is a *Connext Micro* component that can be registered with *Connext Micro* with NETIO_DGRAM_InterfaceFactory_register()_ as shown below:

The factory gets the registry. The registry registers the Datagram.

```
DDS_DomainParticipantFactory *factory = NULL;
RT_Registry_T *registry = NULL;

factory = DDS_DomainParticipantFactory_get_instance();
registry = DDS_DomainParticipantFactory_get_registry(factory);
```

When a component is registered, the registration takes the DGRAM interface as the 3rd parameter and the properties as the 4th parameter. In general, it is up to the caller to manage the memory for the properties and ensure they are valid as long as the DGRAM transport is registered. There is no guarantee that a component makes a copy.

The DGRAM Interface is a component interfaces the *Connext Micro* core library'. The user is responsible for implementing the NETIO_DGRAM_InterfaceI which integrates with a specific network technology. This struct must be compliant with the NETIO_DGRAM_InterfaceI structure.

```
 /* Create the DGRAM User Interface property struct */
 struct MyDgramInterfaceProperty
 {
     RTI_INT32 a_property;
     struct UTEST_Context *setting;
 } MyDgramInterfaceProperty = {10,NULL};

/* Example operation */
 struct NETIO_Interface*
 MyDgramInterface_create_instance(NETIO_Interface_T *upstream,void *property)
 {
     /* Perform operations */
```

(continues on next page)

(continued from previous page)

```
...
   return myInterface;
}
...

/* Create the DGRAM Interface struct where each member points to it's
 * respective operation */
RTI_PRIVATE struct NETIO_DGRAM_InterfaceI MyDgramInterface =
{
    MyDgramInterface_create_instance,
    MyDgramInterface_get_interface_list,
    MyDgramInterface_release_address,
    MyDgramInterface_resolve_address_udpv4,
    MyDgramInterface_send,
    MyDgramInterface_get_route_table,
    MyDgramInterface_bind_address
};
```

The following code snippet shows how to register the DGRAM Interface with new parameters. The Datagram needs to register the DGRAM Interface with a property that has the interface to call:

```
/* Register the transport again, using the builtin name
    */
if (!NETIO_DGRAM_InterfaceFactory_register(registry,
                "name",
                &MyDgramInterface,
                &MyDgramInterfaceProperty))
{
    /* ERROR */
}
```

It should be noted that the Datagram transport can be registered with any name, but all transport QoS policies and initial peers must refer to this name. If a transport is referred to and it does not exist, an error message will be logged.

### 4.11.2 Addressing a Datagram Transport

The interface may also set the enabled transports to receive data as follows:

```
struct DDS_DomainParticipantQos dp_qos =
                                    DDS_DomainParticipantQos_INITIALIZER;

/* Datagram enable transport xyz. A second transport can be added
 * by setting the enabled_transports value to 2 and adding a second
 * transport name. enabled_transport indicates what addresses the entity is
 * listening on.
 */
DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,1);
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
```

(continues on next page)

(continued from previous page)

```
                                        DDS_String_dup("name");

/* Receive discovery traffic on xyz */
DDS_StringSeq_set_maximum(&dp_qos.discovery.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports,1);
*DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports,0) =
                                        DDS_String_dup("name://");

/* Receive user-data traffic on xyz. */
DDS_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports,1);
*DDS_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports,0) =
                                        DDS_String_dup("name://");
```

**Note:** `dp_qos.transports.enabled_transports` refers to a string that contains only a registered transport name, such as the `name` you defined when registering the Datagram transport. `dp_qos.discovery.enabled_transports` and `dp_qos.user_traffic.enabled_transports` require a transport name prefix and (optional) address suffix, as described in *Configuring Discovery Data Reception* and *Configuring User Data Reception*.

An address may setup peers to send messages over this interface. For example, interface xyz may set its initial peers as:

```
 /* Send discovery data on address 0x0A00020F*/
DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers,1);
DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers,1);
*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers,0) =
DDS_String_dup("0x0A00020F");
```

### 4.11.3 Datagram UDP Setup

The built-in Datagram transport can support UDP integration.

Registering the built-in Datagram transport for UDP is different than the generic Datagram component. Use `UDP_Interface_register()` with UDP properties to create the datagram instance for UDP:

```
struct UDP_InterfaceFactoryProperty udp_property =
                                UDP_InterfaceFactoryProperty_INITIALIZER;

/* To enable sharing this property must be set to RTI_FALSE */
udp_property->enable_interface_bind = RTI_TRUE;

/*  */
REDA_StringSeq_set_maximum(&udp_property->allow_interface,1);
REDA_StringSeq_set_length(&udp_property->allow_interface,1);
*REDA_StringSeq_get_reference(&udp_property->allow_interface,0) =
```

(continues on next page)

(continued from previous page)

```
                                                REDA_String_dup(intf);

/* To enable multicast operations the multicast flag and multicast_interface
 * property must be set */
if (is_multicast)
{
    flags |= UDP_INTERFACE_INTERFACE_MULTICAST_FLAG;
    udp_property->multicast_interface = DDS_String_dup(intf);
}
else
{
    /* Set the multicast interface to NULL when not used*/
    udp_property->multicast_interface = NULL;
}

/* Add an available interface for UDP */
if (!UDP_InterfaceTable_add_entry(&udp_property->if_table,
                    address, netmask, intf_name, flags))

{
    /* error */
}

/* Buffer properties */
udp_property->max_send_buffer_size = MAX_SEND_BUFFER_SIZE;
udp_property->max_receive_buffer_size = MAX_RECV_BUFFER_SIZE;
udp_property->max_message_size = MAX_RECV_BUFFER_SIZE;

/* Register the datagram */
if(!UDP_Interface_register(registry, "_udp", udp_property))
{
    /* error */
}
```

Enabled transports can be configured with "_udp://". This will use all interfaces. Enabling UDP is similar to generic addressing:

```
struct DDS_DomainParticipantQos dp_qos =
                                    DDS_DomainParticipantQos_INITIALIZER;

DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,1);
DDS_StringSeq_set_maximum(&dp_qos.discovery.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports,1);
DDS_StringSeq_set_maximum(&dp_qos.user_traffic.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.user_traffic.enabled_transports,1);

/* This only requires the transport name */
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                            DDS_String_dup("_udp");
```

(continues on next page)

(continued from previous page)

```
/* _udp:// indicates to use all available locators */
*DDS_StringSeq_get_reference(&dp_qos.discovery.enabled_transports,0) =
                                        DDS_String_dup("_udp://");

/* _udp://10.10.0.1 would indicate to use only that address */
*DDS_StringSeq_get_reference(&dp_qos.user_traffic.enabled_transports,0) =
                                            DDS_String_dup("_udp://");
```

#### Datagram Shared flag

*RTI Connext Micro* uses Locators to specify transport addresses to send and receive data. A Locator consists of a kind, port, and a transport address. The kind indicates the type of transport, such as UDPv4, the port is used to reach a DDS DomainParticipant and the address is used to reach the destination transport. *RTI Connext Micro* can work with two different types of transports, one that uses shared ports and one that does not.

When a transport uses shared ports it means it does not matter which transport address a message was received on, only the port matters. For example, if a computer has two network interfaces A and B and is listening for messages on port P, it does not matter if the message is received on A or B. That is, as long as the message is received on any network interface capable of receiving on port P, the message is accepted.

When a transport does not use shared ports it means it does matter which transport address a message was received on. For example, if a computer has two network interfaces A and B and is listening for messages on port P, but has only specified that the A should receive on port P, then messages received on interface B and port P are ignored.

*RTI Connext Micro* support this flag on per *RTI Connext Micro* transport basis. It is important to note that when a message is accepted, it is routed to all relevant DDS datareaders and datawriters. Thus, this feature cannot be used to control that some DDS topics should only be accepted when received on a specific transport interface. However, this feature could be useful to allow different DDS DomainParticipants to use the same port, but with different network interfaces.

### 4.11.4 User Interface

NETIO_DGRAM_InterfaceFactory_register() registers a user interface structure that is passed in via `user_intf`. The *DomainParticipant* utilizes these functions for network operations, such as creating a Datagram interface instance and getting the interface list.

Table 4.1: Structure for the User Interface

| Interface Attribute | Description |
|---|---|
| `create_instance` | Creates an instance of the NETIO_DGRAM interface. |
| `delete_instance` | Deletes an instance of the NETIO_DGRAM interface. |
| `get_interface_list` | Reads the available interfaces from the NETIO_DGRAM interface. |

continues on next page

Table 4.1 – continued from previous page

| Interface Attribute | Description |
|---|---|
| `release_address` | Instructs the NETIO_DGRAM interface to stop listening for messages on the source address. |
| `resolve_address` | Instructs the NETIO_DGRAM interface to determine if the address string is valid. |
| `send` | Instructs the NETIO_DGRAM interface to send a message. |
| `get_route_table` | Instructs the NETIO_DGRAM interface `netio_intf` to return a sequence of address and netmask pairs that this interface can send to. |
| `bind_address` | Instructs the NETIO_DGRAM interface to listen for messages on the source address. |

# Chapter 5

# Debugging

## 5.1 Overview

*Connext Micro* maintains a log of events occurring in a *Connext Micro* application. Information on each event is formatted into a log entry. Each entry can be stored in a buffer, stringified into a displayable log message, and/or redirected to a user-defined log handler.

For a list of error codes, please refer to Logging Reference.

## 5.2 Configuring Logging

By default, *Connext Micro* sets the log verbosity to *Error*. It can be changed at any time by calling OSAPI_Log_set_verbosity using the desired verbosity as a parameter.

---

**Note:** Logging is not available in *Connext Cert*.

---

*Connext Micro* stores new log entries in a log buffer.

By default, the Debug libraries are configured with a larger log buffer size than the Release libraries. A custom buffer size can be configured using the OSAPI_Log_set_property function. For example, to set a buffer size of 128 bytes:

```
struct OSAPI_LogProperty prop = OSAPI_LogProperty_INITIALIZER;

OSAPI_Log_get_property(&prop);
prop.max_buffer_size = 128;
OSAPI_Log_set_property(&prop);
```

Note that if the buffer size is too small, log entries will be truncated.

The function used to output the log information can be set by using the function OSAPI_Log_set_property:

```
struct OSAPI_LogProperty prop = OSAPI_LogProperty_INITIALIZER;

OSAPI_Log_get_property(&prop);
prop.write_buffer = <pointer to user defined write function>;
OSAPI_Log_set_property(&prop);
```

The function must have the prototype OSAPI_Log_write_buffer_T and will receive a buffer with the log information formatted as a NUL terminated string.

## 5.3 Log Message Kinds

Each log entry is classified as one of the following kinds:

- *Error*. An unexpected event with negative functional impact.
- *Warning*. An event that may not have negative functional impact but could indicate an unexpected situation.
- *Information*. An event logged for informative purposes.

By default, the log verbosity is set to *Error*, so only error logs will be visible. To change the log verbosity, simply call the function OSAPI_Log_set_verbosity with the desired verbosity level.

## 5.4 Interpreting Log Messages and Error Codes

A log entry in *Connext Micro* has a defined format.

Each entry contains a header with (at minimum) the following information:

- *Module ID*. A numerical ID of the module from which the message was logged.
- *Error Code*. A numerical ID for the log message. It is unique within a module.

---

**Note:** Though referred to as an "error" code, this ID exists for all log kinds (error, warning, info). The module ID and error code together uniquely identify a log message within *Connext Micro*.

---

By default, *Connext Micro* also provides additional details per log message:

- *Module Name*. The name of the module from which the message is logged.
- *File Name*. The name of the file from which the message is logged.
- *Line Number*. The line number of the source file from which the message is logged.
- *Function Name*. The name of the function from which the message is logged.

If desired, you can reduce or disable these details in your log messages with the OSAPI_Log_set_verbosity QoS policy.

**Note:** When the log buffer for error messages fills up, further messages will continue to be logged, but they will only include the *Module ID* and *Error Code*; any additional details will be lost. You can increase the size of your log buffer with the OSAPI_Log_set_property QoS policy.

When an event is logged, by default it is printed as a message to standard output. An example error entry looks like this:

```
[943921909.645099999]ERROR: ModuleID=7 Errcode=200 X=1 E=0 T=1
dds_c/DomainFactory.c:163/DDS_DomainParticipantFactory_get_instance: kind=19
```

Take note of the following error parameters:

- `X` indicates extended debug information is present, such as file and line number.
- `E` indicates an exception; the log message has been truncated.
- `T` indicates the log message has a valid timestamp (e.g., a successful call to `OSAPI_System_get_time()`).

You must interpret the log message when an error or warning occurs and its cause needs to be determined, or if you have set a log handler that processes each log message based on its contents. You can find the description of an error code by following these steps:

1. Go to the Log Codes section of the API Reference.
2. Open the module section corresponding to the *Module Name* in the second line of your error message. If you don't have the *Module Name*, search for the *Module ID* instead. In the example above, `ModuleID=7` corresponds to the DDS C module.
3. Search the module section for the *Error Code*. In the example above, with `Errcode=200`, search for `200`; you will find the log message with the value `DDSC_LOG_BASE + 200`, which is "Out of resources to initialize object of the specified kind."

A description of an error code printed in a log message can be determined by following these steps:

- Navigate to the module that corresponds to the Module ID, or the printed module name in the second line. In the above example, "ModuleID=7" corresponds to DDS.
- Search for the error code to find it in the list of the module's error codes. In the example above, with "Errcode=200," search for "200" to find the log message that has the value "(DDSC_LOG_BASE + 200)".

## 5.5 Writing a Custom Log Handler

*Connext Micro* provides two APIs to override how log events are handled:

- A *display handler*, which displays the log event when called. Install it with OSAPI_Log_set_display_handler. You can use the display handler to show the log information in a suitable format.

- A *log handler*, which processes the log event when called. Install it with OSAPI_Log_set_log_handler. You can use the log handler to store the log event, regardless of how it is displayed.

See the following code snippet for a complete implemenation of a log or display handler:

---

**Note:** This function uses APIs that are not documented. These APIs may change in future releases.

---

```
void
my_log_entry_handler(void *param, OSAPI_LogEntry_T *log_entry)
{
    RTI_INT32 kind;
    RTI_UINT32 eh;
    RTI_UINT32 status = 0;
    const char *module = "<not found>";
    const char *file = "<not found>";
    const char *func = "<not found>";
    RTI_INT32 lineno = -1;
    RTI_UINT32 sec;
    RTI_UINT32 nsec;
    char *data_ptr;
    OSAPI_LogType_T type;
    RTI_BOOL is_final = RTI_FALSE;
    const char *name;
    const void *value;

    UNUSED_ARG(param);

    /* kind is one of the following:
     * OSAPI_LOGKIND_INFO
     * OSAPI_LOGKIND_WARNING
     * OSAPI_LOGKIND_ERROR
     * OSAPI_LOGKIND_PRECONDITION
     */
    kind = OSAPI_LOG_HEADER_GET_TYPE(log_entry->error_code);

    /* Extract detailed information from the event header. If the
     * X-bit is set, it means extended information is available.
     */
    eh = log_entry->error_code;
    if (OSAPI_LOG_HEADER_GET_X(log_entry->error_code))
    {
        data_ptr = (char*)(log_entry + 1);
        status = *OSAPI_Compiler_reinterpret_cast(RTI_UINT32*,data_ptr);
        data_ptr += sizeof(RTI_INT32);
        if (status & OSAPI_LOG_STATUS_LN)
        {
            lineno = *OSAPI_Compiler_reinterpret_cast(RTI_INT32*,data_ptr);
            data_ptr += (RTI_INT32)sizeof(RTI_INT32);
        }
```

(continues on next page)

(continued from previous page)

```
        if (status & OSAPI_LOG_STATUS_MN)
        {
            module = data_ptr;
            data_ptr += OSAPI_String_length(data_ptr) + 1;
        }
        if (status & OSAPI_LOG_STATUS_SF)
        {
            file = data_ptr;
            data_ptr += OSAPI_String_length(data_ptr) + 1;
        }
        if (status & OSAPI_LOG_STATUS_FN)
        {
            func = data_ptr;
            data_ptr += OSAPI_String_length(data_ptr) + 1;
        }
        if (status & OSAPI_LOG_STATUS_F)
        {
            data_ptr += OSAPI_String_length(data_ptr) + 1;
        }
    }

    sec = (RTI_UINT32)log_entry->timestamp.sec;
    nsec = log_entry->timestamp.nanosec;

    /* At this point the followig information is available:
     *  module - The module the event occured in
     *  file - The file the event occured in
     *  func - The function the event occured in
     *  lineno - The line number in file the event occured in
     *  sec - The seconds the event occurred
     *  nsec - The nanoseconds the event occurred
     */

    /* There may be additional information available, extract each
     * individual field.
     */
    data_ptr = NULL;
    is_final = RTI_FALSE;
    while (OSAPI_Log_entry_get_data(log_entry,&data_ptr,&type,
                                    &name,&value,&is_final))
    {
        /* On successful return, value points to a value of the specified
         * type
         */
        switch (type)
        {
            case OSAPI_LOGTYPE_HEX:
                /* value points to a 32-bit integer that is expected to
                 * be displayed in hex format.
                 */
                break;
```

(continues on next page)

(continued from previous page)

```
            case OSAPI_LOGTYPE_INTEGER:
                /* value points to a 32-bit integer that is expected to
                 * be displayed in decimal format.
                 */
                break;
            case OSAPI_LOGTYPE_UINTEGER:
                /* value points to a 32-bit unsigned integer that is
                 * expected to be displayed in decimal format.
                 */
                break;
            case OSAPI_LOGTYPE_POINTER:
                /* value points to a pointer.
                 */
                break;
            case OSAPI_LOGTYPE_STRING:
                /* value points to a NULL terminated string.
                 */
                break;
            default:
                break;
        }

        if (is_final)
        {
            /* there are no more values to display, exit loop */
            break;
        }
    }
}
```

## 5.6 Contact Support

We welcome your input on how to improve *RTI Connext Micro* to suit your needs. If you have questions or comments about this release, please visit the RTI Customer Portal, https://support.rti.com. The RTI Customer Portal provides access to RTI software, documentation, and support. It also allows you to log support cases.

To access the software, documentation or log support cases, the RTI Customer Portal requires a username and password. You will receive this in the email confirming your purchase. If you do not have this email, please contact license@rti.com. Resetting your login password can be done directly at the RTI Customer Portal.

## 5.7 Join the Community

RTI Community provides a free public knowledge base containing how-to guides, detailed solutions, and example source code for many use cases. Search it whenever you need help using and developing with RTI products.

RTI Community also provides forums for all RTI users to connect and interact.

# Chapter 6

# Platforms Guide

## 6.1 Introduction

This section provides platform-specific instructions that you will need to build and run *RTI Connext Micro* applications.

For each supported operating system (OS), this section describes:

- Supported combinations of OS versions, CPUs, and compilers
- How to build your application, including:
  - Required *Connext Micro* and system libraries
  - Required compiler and linker flags
  - Details on how the *Connext Micro* libraries were built

To see a list of all supported platforms, refer to *Supported Platforms and Programming Languages*.

### 6.1.1 Library types

*Connext Micro* provides precompiled binaries for supported architectures. This section explains the different library types and gives a general description of the binaries shipped by RTI.

In this section, the following terms are used:

- *toolchain* refers to the compiler, linker, and archiver for a specific CPU architecture, excluding dependencies in standard header files and libraries.
- *platform* refers to the hardware, BSPs, OS kernel, and C/C++ libraries that are not included in the toolchain (such as `libc`, `libc++`, and the network stack).

RTI builds two types of binaries for *Connext Micro*: integrated libraries and split libraries. RTI may include either or both types of binaries for a given target architecture.

### Integrated libraries

All *Connext Micro* target packages include a core library called `rti_me`. The `rti_me` library includes all the required basic functionality for *Connext Micro*.

The term "integrated library" refers to an `rti_me` library where all the OS integration and network stack integration is compiled directly into `rti_me`. This means that it is not possible to change how the OS and network integration has been written without recompiling the entire library. This is illustrated below:

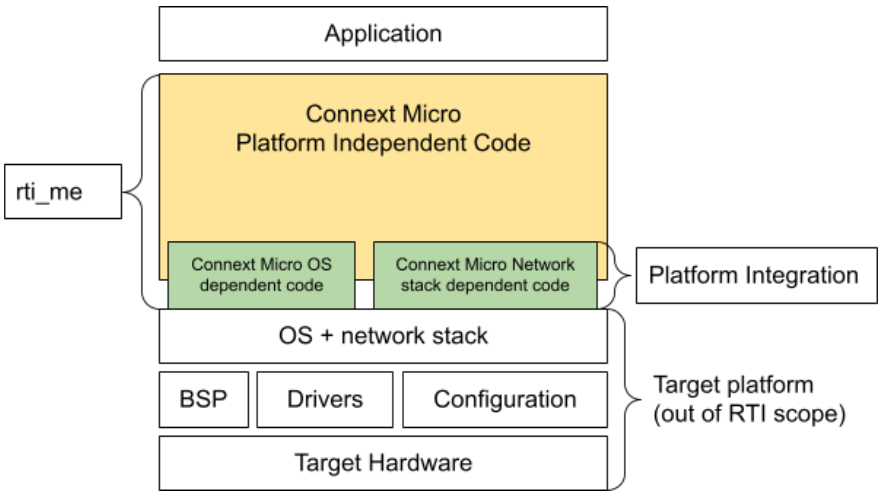



Figure 6.1: An overview of rti_me as an integrated library

**Note:** All binaries provided for *Connext Micro* version 4.0.1 and below are integrated libraries.

### Split libraries

In contrast to an integrated library, split libraries consist of a Platform Independent Library (PIL) and a Platform Support Library (PSL).

The PIL is an `rti_me` library that includes all functionality for *Connext Micro* except for platform integration code.

The PSL consists of two libraries that support OS integration and network stack integration:

- The OS Platform Support Library (`ospsl`): Contains the required OS support, such as mutex and semaphore support. This library is very limited in functionality.
- The Network Support Library (`netiopsl`): Includes support for transports, such as UDPv4.

The `ospsl` and `netiopsl` libraries are collectively referred to as the PSL (even though it is more than one library).

This is illustrated below:

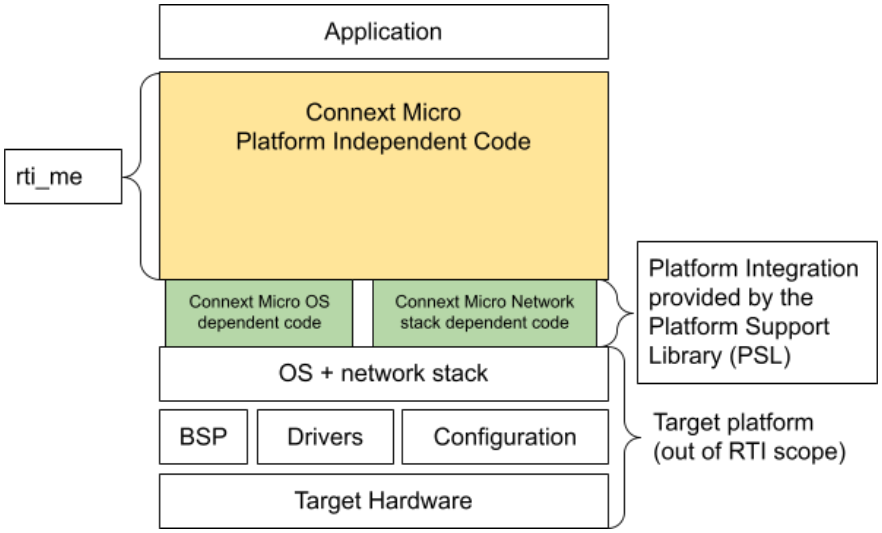



Figure 6.2: An overview of split libraries

The main benefit of split libraries is that different PSLs can be written for the same PIL without having to recompile the platform-independent code.

---

**Note:** The PIL is compiled without standard C header files and is only dependent on the toolchain. This is different from the integrated libraries, which are compiled with standard C header files.

The PSL is always compiled against the standard C header files, as well as other platform-dependent header files.

---

### 6.1.2 Library descriptions

The following libraries are included in the target bundle. Note that the names listed below do not include platform-specific prefixes or suffixes.

Depending on the target architecture, the library name is prefixed with lib and the library suffix also varies between target architectures, such as .so and .dylib.

The following naming conventions are also used:

- Static libraries have a z suffix.
- Shared libraries do not have an additional suffix.
- Debug libraries have a d suffix.
- Release libraries do not have an additional suffix.

For example:

- `rti_mezd` indicates a static debug library.
- `rti_me` indicates a dynamically linked release library.

Table 6.1: Target Bundle Libraries

| Library Name | Description |
|---|---|
| `rti_me` | The core library, including the DDS C API. |
| `rti_me_discdpse` | The Dynamic Participant Static Endpoint (DPSE) plugin. |
| `rti_me_discdpde` | The Dynamic Participant Dynamic Endpoint (DPDE) plugin. |
| `rti_me_rhsm` | The Reader History plugin. |
| `rti_me_whsm` | The Writer History plugin. |
| `rti_me_netiosdm` | The Zero Copy v1 over shared memory transport library plugin. |
| `rti_me_netioshmem` | The Shared Memory Transport plugin. |
| `rti_me_appgen` | The Application Generation plugin. |
| `rti_me_cpp` | The C++ API. |
| `rti_me_ospsl` | The OS PSL. |
| `rti_me_netiopsl` | The C NETIO PSL. |
| `rti_me_netiopsl_cpp` | The C++ NETIO PSL library. |
| `rti_me_netiozcopy` | The Zero Copy v2 transport library plugin (not supported on all platforms). |
| `rti_me_ddsfilter` | The content filtering library. |
| `rti_me_ddsxtypes` | The XTypes library. |
| `rti_me_ddspsk` | The Lightweight Security Plugin PIL (installed separately, not supported on all platforms). |
| `rti_me_pskpsl_openssl3` | The Lightweight Security Plugin Transform PSL (installed separately, not supported on all platforms). |

### 6.1.3 Build profiles

You can optionally build *Connext Micro* with a CERT profile. This restricts *Connext Micro* to only include features that are available or planned for *Connext Cert*; for more information, see *Building Connext Micro with Compatibility for Connext Cert*.

Any architecture ending with CERT is built with the CERT profile enabled.

Some features are only available on specific platforms; see the footnotes in the table below.

Table 6.2: Features by Profile

| Feature/Capability | Non-CERT Profile | CERT Profile |
|---|---|---|
| Dynamic Participant Discovery | ✓ | ✓ |
| Static Endpoint Discovery | ✓ | ✓ |
| Dynamic Endpoint Discovery | ✓ | |
| C++ API | ✓[1] | |
| Shared Memory Transport (SHMEM) | ✓[1] | |
| Zero Copy v1 | ✓[1] | |
| Zero Copy v2 | ✓[2] | ✓[2] |
| Micro Application Generator (MAG) | ✓ | |
| Content Filtering | ✓ | ✓[3] |
| Lightweight Security Plugin | ✓[4] | ✓[4] |
| XTypes | ✓ | |

### 6.1.4 Supported libraries by platform

The following table shows which *Connext Micro* libraries are supported on each platform (RTI architecture).

Table 6.3: Supported Libraries by Platform

| Platform | RTI Architecture | Supported Libraries |
|---|---|---|
| Windows 10 x64 | x86_64lePEvs2017 | `rti_me`<br>`rti_me_whsm`<br>`rti_me_rhsm`<br>`rti_me_discdpse`<br>`rti_me_discdpde`<br>`rti_me_netiosdm`<br>`rti_me_netioshmem`<br>`rti_me_appgen`<br>`rti_me_cpp`<br>`rti_me_ddsxtypes`<br>`rti_me_ddsfilter`<br>`rti_me_ddspsk` |
| | x86_64lePEvs2017CERT | `rti_me`<br>`rti_me_ddsfilter`<br>`rti_me_ddspsk` |
| macOS 14 arm64 | armv8leMachOclang15.0 | `rti_me`<br>`rti_me_whsm`<br>`rti_me_rhsm`<br>`rti_me_discdpse`<br>`rti_me_discdpde`<br>`rti_me_netiosdm`<br>`rti_me_netioshmem`<br>`rti_me_appgen`<br>`rti_me_cpp`<br>`rti_me_ddsxtypes`<br>`rti_me_ddsfilter` |
| | armv8leMachOclang15.0CERT | `rti_me`<br>`rti_me_ddsfilter` |

continues on next page

[1] Not supported on FreeRTOS or MicroSAR platforms.
[2] Not supported on FreeRTOS, macOS, MicroSAR, or Windows platforms.
[3] The built-in SQL filter is not supported with the CERT profile. Only custom content filters are supported.
[4] Not supported on FreeRTOS, macOS, or MicroSAR platforms.

Table 6.3 – continued from previous page

| Platform | RTI Architecture | Supported Libraries |
|---|---|---|
| Ubuntu 24.04 x64 | x86_64leElfgcc13.3.0 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_netiosdm<br>rti_me_netioshmem<br>rti_me_netiozcopy<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| | x86_64leElfgcc13.3.0CERT | rti_me<br>rti_me_netiozcopy<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| Ubuntu 24.04 ARMv8 | armv8leElfgcc13.3.0 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_netiosdm<br>rti_me_netioshmem<br>rti_me_netiozcopy<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| | armv8leElfgcc13.3.0CERT | rti_me<br>rti_me_netiozcopy<br>rti_me_ddsfilter<br>rti_me_ddspsk |

continues on next page

Table 6.3 – continued from previous page

| Platform | RTI Architecture | Supported Libraries |
|---|---|---|
| Yocto 5.15.96 | armv8aleElfgcc11.3.1 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_netiosdm<br>rti_me_netioshmem<br>rti_me_netiozcopy<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| | armv8aleElfgcc11.3.1CERT | rti_me<br>rti_me_netiozcopy<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| QNX 7.1 ARMv8 | armv8leElfqnx_qcc8.3.0 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_netiosdm<br>rti_me_netioshmem<br>rti_me_netiozcopy<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| | armv8leElfqnx_qcc8.3.0CERT | rti_me<br>rti_me_netiozcopy<br>rti_me_ddsfilter<br>rti_me_ddspsk |

continues on next page

Table 6.3 – continued from previous page

| Platform | RTI Architecture | Supported Libraries |
|---|---|---|
| QNX 8.0 ARMv8 | armv8leElfqnx_qcc12.2.0 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_netiosdm<br>rti_me_netioshmem<br>rti_me_netiozcopy<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| | armv8leElfqnx_qcc12.2.0CERT | rti_me<br>rti_me_netiozcopy<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| QOS 2.2.1 (QNX OS for Safety) | armv8leElfqcc8.3.0 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_netiosdm<br>rti_me_netioshmem<br>rti_me_netiozcopy<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| | armv8leElfqcc8.3.0CERT | rti_me<br>rti_me_netiozcopy<br>rti_me_ddsfilter<br>rti_me_ddspsk |
| FreeRTOS 10.0.0 ARMv7E-M | armv7emleElfgcc10.3.1 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_appgen<br>rti_me_cpp<br>rti_me_ddsxtypes<br>rti_me_ddsfilter |

continues on next page

Table 6.3 – continued from previous page

| Platform | RTI Architecture | Supported Libraries |
|---|---|---|
| | armv7em-leElfgcc10.3.1CERT | rti_me<br>rti_me_ddsfilter |
| MicroSAR 4.2.2 | tc39xtElfTasking6.3r1 | rti_me<br>rti_me_whsm<br>rti_me_rhsm<br>rti_me_discdpse<br>rti_me_discdpde<br>rti_me_appgen<br>rti_me_ddsxtypes<br>rti_me_ddsfilter |
| | tc39xtElfTask-ing6.3r1CERT | rti_me<br>rti_me_ddsfilter |

### 6.1.5 Supported transports by platform

The following table shows which transports are supported on each architecture.

Table 6.4: Supported Transports by Platform

| Platform | RTI Architecture | Intra | UDPv4 | SHMEM | Zero Copy v1 | Zero Copy v2 |
|---|---|---|---|---|---|---|
| Windows 10 x64 | x86_64lePEvs2017 | ✓ | ✓ | ✓ | ✓ | |
| | x86_64lePEvs2017CERT | ✓ | ✓ | | | |
| macOS 14 arm64 | armv8leMachOclang15.0 | ✓ | ✓ | ✓ | ✓ | |
| | armv8leMachOclang15.0CERT | ✓ | ✓ | | | |
| Ubuntu 24.04 x64 | x86_64leElfgcc13.3.0 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | x86_64leElfgcc13.3.0CERT | ✓ | ✓ | | | ✓ |
| Ubuntu 24.04 ARMv8 | armv8leElfgcc13.3.0 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | armv8leElfgcc13.3.0CERT | ✓ | ✓ | | | ✓ |
| Auto Yocto Linux BSP 37.0 | armv8aleElfgcc11.3.1 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | armv8aleElfgcc11.3.1CERT | ✓ | ✓ | | | ✓ |
| QNX 7.1 ARMv8 | armv8leElfqnx_qcc8.3.0 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | armv8leElfqnx_qcc8.3.0CERT | ✓ | ✓ | | | ✓ |
| QNX 8.0 ARMv8 | armv8leElfqnx_qcc12.2.0 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | armv8leElfqnx_qcc12.2.0CERT | ✓ | ✓ | | | ✓ |
| QOS 2.2.1 (QNX OS for Safety) | armv8leElfqcc8.3.0 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | armv8leElfqcc8.3.0CERT | ✓ | ✓ | | | ✓ |
| FreeR-TOS 10.4.6 ARMv7E-M | armv7emleElfgcc10.3.1 | ✓ | ✓ | | | |
| | armv7emleElfgcc10.3.1CERT | ✓ | ✓ | | | |
| MicroSAR 4.2.2 | tc39xtElfTasking6.3r1 | ✓ | ✓ | | | |
| | tc39xtElfTasking6.3r1CERT | ✓ | ✓ | | | |

## 6.2 Building Connext Micro for Common Platforms

This section describes how to compile *Connext Micro*, either the Platform Support Libraries (PSL) or the full source if available, for an architecture supported by RTI (see *Supported Platforms and Programming Languages* for more information).

For information about how to compile and link *Connext Micro* applications, please refer to *Prepare Your Development Environment*.

This section is written for developers and engineers with a background in software development. RTI recommends reading this section in order, as one subsection may refer to or assume knowledge about concepts described in a preceding subsection.

### 6.2.1 Setting up the build environment

The following terminology is used to refer to the environment in which *Connext Micro* is built and run:

- The *host* is the machine that runs the software to compile and link *Connext Micro*.
- The *target* is the machine that will run *Connext Micro*.
- In many cases *Connext Micro* is built *and* run on the same machine. This is referred to as a *self-hosted environment*.

The *environment* is the collection of tools, OS, compiler, linker, hardware etc. needed to build and run applications.

The word *must* describes a requirement that must be met. Failure to meet a *must* requirement may result in failure to compile, use, or run *Connext Micro*.

The word *should* describes a requirement that is strongly recommended to be met. A failure to meet a *should* recommendation may require modification to how *Connext Micro* is built, used, or run.

The word *may* is used to describe an optional feature.

#### The host environment

*Connext Micro* has been designed to be easy to build and to require few tools on the host.

The host machine **must**:

- support long filenames (8.3 will not work). *Connext Micro* does not require a case-sensitive file-system.
- have the necessary compiler, linkers, and build-tools installed.

The host machine **should**:

- have CMake installed. Note that it is not required to use CMake to build *Connext Micro*, but as a rule of thumb, if you are comfortable building *Connext Micro* from the command-line, we recommend using CMake.

- be able to run bash shell scripts (Linux and macOS systems) or BAT scripts (Windows systems).

Supported host environments are Windows (cygwin and mingw are not tested), Linux, and macOS systems.

Typical examples of host machines are:

- a Linux PC with the GNU tools installed (make, gcc, g++, etc.).
- a Mac computer with Xcode and the command-line tools installed.
- a Windows computer with Microsoft Visual Studio Express edition.
- a Linux, Mac or Windows computer with an embedded development tool-suite.

#### The target environment

The target compiler should:

- be C99 compliant. Note that many non-standard compilers work, but may require additional configuration.
- be C++98 compliant.

The remainder of this manual assumes that the target environment is one supported by RTI:

- POSIX (Linux, macOS, QNX®)
- Windows

### 6.2.2 Building the PSL

As described in *Library descriptions*, *Connext Micro* comes with Platform Support Libraries (PSL) that are compatible with specific Platform Independent Libraries (PIL). The source code for the PSL is also available in the host package. This section describes how to build the PSL from the provided source code.

RTI provides the PSL source code because it allows the PSL to be recompiled for a specific platform configuration. This may be important in some use cases if the header files include platform-specific information that is different from the binaries provided by RTI.

There are two recommended methods to compile the PSL: by running the rtime-make script (which invokes CMake), or by invoking CMake manually. Both are described in more detail below.

CMake is the preferred tool to build *Connext Micro* because it simplifies configuring the *Connext Micro* build options and generates build files for a variety of environments. Note that CMake itself does not compile anything. CMake is used to *generate* build files for a number of environments, such as make, Eclipse® CDT, Xcode® and Visual Studio. Once the build-files have been generated, any of the tools mentioned can be used to build *Connext Micro*. This system makes it easier to support building *Connext Micro* in different build environments. CMake is easy to install with pre-built binaries for common environments and does not depend on external tools to build *Connext Micro*.

### Building the PSL with rtime-make

The *Connext Micro* source bundle includes a bash (on Linux and macOS systems) and BAT (on Windows systems) script to simplify the invocation of CMake. These scripts are a convenient way to invoke CMake with the correct options.

---

**Note:** rtime-make must be invoked from the RTIMEHOME directory.

---

Run the rtime-make script with the following command:

Linux

```
RTIMEHOME/resource/scripts/rtime-make --config Debug --target x86_64leElfgcc7.3.0-Linux4␣
↪\
                        -G "Unix Makefiles" --build
```

Windows

```
RTIMEHOME\resource\scripts\rtime-make -A x64 --config Debug --target x86_64lePEvs2017-
↪Win10 \
                    -G "Visual Studio 15 2017" --build
```

When the compilation has finished, the PSL is copied to the directory `RTIMEHOME/lib/<target>` where `<target>` is the argument passed to the `--target <target>`.

> **Warning:** The above command will overwrite the PSL installed by RTI. To use a different output directory refer to *Specifying a different output directory*.

Here is an explanation of each argument in the above command:

- `--config Debug`: Create Debug build. Use `--config Release` to create a release build.
- `--target <target>`: The target for the sources to be built. Refer to *Supported Platforms and Programming Languages* for the architecture abbreviations of supported platforms.
- `--build`: Build the generated project files.

To get a list of all the options, run:

```
rtime-make -h
```

To get help for a specific target, run:

```
rtime-make --target <target> --help
```

#### Specifying a different output directory

By default, rtime-make copies the compiled PSL into a directory named `RTIMEHOME/lib/<target>` where `<target>` is the argument that was passed to the `--target <target>` option.

To copy the compiled PSL to a different output directory, the `--name <name>` option can be used together with `--target <target>`. In this case, the PSL will be compiled using the same options as specified for `--target <target>`, but instead the PSLs will be copied to the directory `RTIMEHOME/lib/<name>`.

---

**Note:** You should use the same naming convention for `--name` as for `--target`. *Connext Micro* may use the directory name to determine the appropriate Platform Independent Library (PIL) for the compiled PSL.

---

For example, the following command will compile the PSL using the same target configuration as for `x86_64leElfgcc7.3.0-Linux4`, but copy the compiled PSL to `RTIMEHOME/lib/x86_64leElfgcc7.3.0-mypsl`

```
RTIMEHOME/resource/scripts/rtime-make --config Debug \
    --target x86_64leElfgcc7.3.0-Linux4 \
    --name x86_64leElfgcc7.3.0-mypsl \
    -G "Unix Makefiles" --build
```

### Building the PSL with CMake

#### Preparing to build

RTI recommends creating a unique directory for each build configuration. A build configuration can be created to address specific architectures, compiler settings, or different *Connext Micro* build options.

RTI recommends assigning a descriptive *name* to each build configuration, using a common format. While there are no requirements to the format for functional correctness, the toolchain files in *Cross-compiling Connext Micro* use the `<name>` parameter passed to `--target <name>` to determine various compiler options and selections.

RTI uses the following format for the target architecture PSL:

```
{cpu}{compiler}{profile}-{OS}
```

- `{cpu}`: the CPU that the library was compiled for.
- `{compiler}`: the compiler used to build the library.
- `{profile}`: CERT if the library was built to be Cert-compatible; otherwise empty.
- `{OS}`: The operating system that the PSL was compiled for.

Some examples of target names:

- `x86_64leElfgcc7.3.0-Linux4`: PSL for *Connext Micro* for an x64 CPU compiled using GCC 7.3.0 and running a Linux4 kernel.
- `x86_64lePEvs2017-Win10`: PSL for *Connext Micro* for an x64 CPU compiled using VS2017 and running in Windows 10.

Files built by each build configuration will be stored under `RTIMEHOME/build/[Debug | Release]/<name>`. These directories are referred to as build directories or `RTIMEBUILD`. The structure of the `RTIMEBUILD` depends on the generated build files and should be regarded as an intermediate directory.

### Creating build files from the command line

Open a terminal window in the `RTIMEHOME` directory and create the `RTIMEBUILD` directory. Change to the `RTIMEBUILD` directory and invoke CMake with the following arguments:

**Note:** This section assumes that cmake is invoked from the RTIMEHOME directory.

```
cmake -G <generator> -DCMAKE_BUILD_TYPE=<Debug | Release> \
      -DCMAKE_TOOLCHAIN_FILE=<toolchain file>  \
      -DCMAKE_MODULE_PATH=RTIMEHOME/resource/cmake/architectures \
      -DRTIME_TARGET_NAME=<target-name> \
      -DRTIME_TARGET=<target-name> \
      RTIMEHOME
```

Depending on the generator, do one of the following:

- For IDE generators (such as Eclipse, Visual Studio, Xcode), open the generated solution/project files and build the project/solution.
- For command-line tools (such as make, nmake, ninja), run the build-tool.

After a successful build, the output is placed in `RTIMEHOME/lib/<target-name>`.

The generated build files may contain different sub-projects that are specific to the tool. For example, in Xcode and Visual Studio, the following targets are available:

- `ALL_BUILD`: Builds all the projects.
- `\rti_me_<name>`: Builds only the specific library. Note that dependent libraries are built first.
- `ZERO_CHECK`: Runs CMake to regenerate project files in case something changed in the build input. This target does not need to be built manually.

For command-line tools, try `<tool> help` for a list of available targets to build. For example, if makefiles were generated:

```
make help
```

### 6.2.3 Building the source

> **Warning:** This section only applies to *Connext Micro* source bundles (`rti_connext_dds_micro-<version>-source.zip`). For other bundles, refer to *Building the PSL*.

When you build *Connext Micro* from the source bundle, you have two options:

- Build a Platform Independent Library (PIL) and a compatible Platform Support Library (PSL).
- Build an integrated library.

The source code for the PIL, PSL, and integrated libraries are all included in the source bundle. Refer to *Library descriptions* for more information on the differences between them. The following sections explain how to compile each library type.

There are two recommended methods to compile libraries: by running the rtime-make script (which invokes CMake), or by invoking CMake manually. Both are described in more detail below.

#### Building with rtime-make

The *Connext Micro* source bundle includes a bash (on Linux and macOS systems) and BAT (on Windows systems) script to simplify the invocation of CMake. These scripts are a convenient way to invoke CMake with the correct options.

Run the rtime-make script with the following commands:

Linux

To build the PIL:

```
RTIMEHOME/resource/scripts/rtime-make --config Debug --target x86_64leElfgcc7.3.0 \
                      -G "Unix Makefiles" --build
```

To build the PSL using the above PIL:

```
RTIMEHOME/resource/scripts/rtime-make --config Debug --target x86_64leElfgcc7.3.0-Linux4␣
↪\
                      -G "Unix Makefiles" --build
```

To build the integrated library:

```
RTIMEHOME/resource/scripts/rtime-make --config Debug --target x64Linux4gcc7.3.0 \
                      -G "Unix Makefiles" --build
```

Windows

To build the PIL:

```
RTIMEHOME\resource\scripts\rtime-make -A x64 --config Debug --target x86_64lePEvs2017 \
                  -G "Visual Studio 15 2017" --build
```

To build the PSL using the above PIL:

```
RTIMEHOME\resource\scripts\rtime-make -A x64 --config Debug --target x86_64lePEvs2017-
↪Win10 \
                  -G "Visual Studio 15 2017" --build
```

To build the integrated library:

```
RTIMEHOME\resource\scripts\rtime-make -A x64 --config Debug --target x64Win64VS2017 \
                  -G "Visual Studio 15 2017" --build
```

Here is an explanation of each argument in the above commands:

- `--config Debug`: Create Debug build.
- `--target <target>`: The target for the sources to be built. Refer to *Supported Platforms and Programming Languages* for the architecture abbreviations of supported platforms.
- `--build`: Build the generated project files.

To get a list of all the options, run:

```
rtime-make -h
```

To get help for a specific target, run:

```
rtime-make --target <target> --help
```

#### Building with CMake

#### Preparing to build

RTI recommends creating a unique directory for each build configuration. A build configuration can be created to address specific architectures, compiler settings, or different *Connext Micro* build options.

RTI recommends assigning a descriptive *name* to each build configuration, using a common format. While there are no requirements to the format for functional correctness, the toolchain files in *Cross-compiling Connext Micro* use the **RTIME_TARGET_NAME** variable to determine various compiler options and selections.

RTI uses the following formats for the target architecture libraries:

PIL

```
{cpu}{compiler}{profile}
```

- `{cpu}`: the CPU that the library was compiled for.

- {compiler}: the compiler used to build the library.
- {profile}: CERT if the library was built to be Cert-compatible; otherwise empty.

PSL

```
{cpu}{compiler}{profile}-{OS}
```

- {cpu}: the CPU that the library was compiled for.
- {compiler}: the compiler used to build the library.
- {profile}: CERT if the library was built to be Cert-compatible; otherwise empty.
- {OS}: The operating system that the PSL was compiled for.

Integrated

```
{cpu}{OS}{compiler}{profile}
```

- {cpu}: the CPU that the library was compiled for.
- {OS}: The operating system that the integrated library was compiled for.
- {compiler}: the compiler used to build the library.
- {profile}: CERT if the library was built to be Cert-compatible; otherwise empty.

Some examples of target names:

- x86_64leElfgcc7.3.0: PIL for an x64 CPU, compiled with gcc 7.3.0.
- x86_64leElfgcc7.3.0-Linux4: PSL for an x64 CPU, running Ubuntu 18.04 LTS, compiled with gcc 7.3.0.
- x64Linux4gcc7.3.0: Integrated library for an x64 CPU, running Ubuntu 18.04 LTS, compiled with gcc 7.3.0.
- x86_64lePEvs2017: PIL for an x64 CPU, compiled with Visual Studio 2017.
- x86_64lePEvs2017-Win10: PSL for an x64 CPU, running Windows 10, compiled with Visual Studio 2017.
- x64Win64VS2017: Integrated library for an x64 CPU, running Windows 10, compiled with Visual Studio 2017.

Files built by each build configuration will be stored under RTIMEHOME/build/[Debug | Release]/<name>. These directories are referred to as build directories or RTIMEBUILD. The structure of the RTIMEBUILD depends on the generated build files and should be regarded as an intermediate directory.

### Creating build files from the command line

**Note:** This section assumes that CMake is invoked from the `RTIMEHOME` directory. For out-of-source builds using CMake, refer to *Building with CMake outside of source*.

Open a terminal window in the `RTIMEHOME` directory and create the `RTIMEBUILD` directory. Change to the `RTIMEBUILD` directory and invoke CMake with the following arguments:

```
cmake -G <generator> -DCMAKE_BUILD_TYPE=<Debug | Release> \
      -DCMAKE_TOOLCHAIN_FILE=<toolchain file>  \
      -DCMAKE_MODULE_PATH=RTIMEHOME/resource/cmake/architectures \
      -DRTIME_TARGET_NAME=<target-name> \
      -DRTIME_TARGET=<target-name> \
      RTIMEHOME
```

Depending on the generator, do one of the following:

- For IDE generators (such as Eclipse, Visual Studio, Xcode), open the generated solution/project files and build the project/solution.
- For command-line tools (such as make, nmake, ninja), run the build-tool.

After a successful build, the output is placed in `RTIMEHOME/lib/<name>`.

The generated build-files may contain different sub-projects that are specific to the tool. For example, in Xcode and Visual Studio the following targets are available:

- `ALL_BUILD`: Builds all the projects.
- `\rti_me_<name>`: Builds only the specific library. Note that that dependent libraries are built first.
- `ZERO_CHECK`: Runs CMake to regenerate project files in case something changed in the build input. This target does not need to be built manually.

For command-line tools, try `<tool> help` for a list of available targets to build. For example, if makefiles were generated:

```
make help
```

### Building with CMake outside of source

**Note:** This option is only available with the *Connext Micro* source bundle.

You may need to build *Connext Micro* in a directory that is located outside `RTIMEHOME` and to install *Connext Micro* in a separate installation directory. In this case, do the following:

1. Create a build directory and change the current directory to it.

2. Invoke CMake with the following command to create build files for an integrated library:

```
cmake -DCMAKE_TOOLCHAIN_FILE=<RTI toolchain> \
      -DRTIME_TARGET=<target-name> \
      -DRTIME_TARGET_NAME=<target-name> \
      -DCMAKE_BUILD_TYPE=[debug | Release]  \
      <path to RTIMEHOME/CMakeLists.txt>
```

   Alternatively, you can create build files for split libraries. This requires two commands, one for the PIL and one for the PSL.

   To build the PIL:

```
cmake -DCMAKE_TOOLCHAIN_FILE=<RTI toolchain> \
      -DRTIME_TARGET=<target-name> \
      -DRTIME_TARGET_NAME=<target-name> \
      -DCMAKE_BUILD_TYPE=[debug | Release]  \
      <path to RTIMEHOME/CMakeLists.txt>
```

   To build the PSL:

```
cmake -DCMAKE_TOOLCHAIN_FILE=<RTI toolchain> \
      -DRTIME_TARGET=<target-name> \
      -DRTIME_TARGET_NAME=<target-name> \
      -DCMAKE_BUILD_TYPE=[debug | Release]  \
      -DRTIME_PIL_PATH=<path to  RTI's PIL if compiling a PSL> \
      <path to RTIMEHOME/CMakeLists.txt>
```

3. Invoke the build tool on the generated build-files:

```
make
```

4. Install the compiled libraries using `cmake --install`:

```
cmake --install . --prefix <install path>
```

The last command will copy the header files and libraries to:

- `<install path>/include` (header files)
- `<install path>/lib` (libraries)

### 6.2.4 Cross-compiling Connext Micro

Cross-compiling the *Connext Micro* source-code uses the exact same process described in *Building the source*, but requires an additional toolchain file. A toolchain file is a CMake file that describes the compiler, linker, etc., needed to build the source for the target.

To see a list of available targets, use `--list`:

```
rtime-make --list
```

## 6.3 Building Connext Micro with Compatibility for Connext Cert

*RTI Connext Micro* can be compiled to only include features that are available or planned for *RTI Connext Cert*. This is useful to enable the development of a safety-certified project using *Connext Micro* before certification evidence for *Connext Cert* is available. Once *Connext Cert* certification is available, the transition from *Connext Micro* to *Connext Cert* typically requires few changes in the application. The *Connext Micro* Cert-compatibility profile refers to the subset of *Connext Micro* that is feature-comparable to *Connext Cert*.

**Warning:** Please note that this does not mean that certification evidence is provided for *Connext Micro* for any of these features or that using the Cert-compatibility profile constitutes safety. Please contact RTI for further information about *Connext Cert* and certification evidence.

**Warning:** The Dynamic Participant, Dynamic Endpoint (DPDE) discovery plugin is available with the CERT profile for convenience during initial development. However, please note the following:

- The DPDE plugin is not part of the Connext Cert roadmap. RTI will not certify this library.
- The DPDE library does not support restarts as it depends on functionality not included in the CERT profile's core library.

When compiling *Connext Micro* with compatibility for *Connext Cert*, please refer to the *Platform Notes* for information on supported platforms, libraries, and features in CERT profiles. The following additional restrictions apply:

- The CERT profile `rti_me` library **includes** the following libraries:
  - `rti_me_discdpse`
  - `rti_me_rhsm`
  - `rti_me_whsm`
  - `rti_me_discdpde`
- Memory deallocation is not possible
- Any API that deallocates memory is not supported. In other words, any API whose name includes "finalize", "free", or "delete" is not supported (such as `DDS_DomainParticipantFactory_delete_participant()`, `DDS_DomainParticipantQos_finalize()`, or `OSAPI_Heap_free()`)
- Code generated by the *Connext Micro* code generator is compatible with *Connext Cert*, but the code must be generated with the code generator using the `-interpreted 0` option
- The Log module is only available in the debug build

- The UDP transport *must* be configured statically by using the API `UDP_InterfaceTable_add_entry()` and setting `UDP_InterfaceFactoryProperty.disable_auto_interface_config` equal to `RTI_TRUE`
- `OSAPI_Thread_sleep()` is not available
- Batching reception is not supported
- UDP Transformations are not supported
- The Property, User Data, and Partition Qos APIs are available in the *Connext Micro* Cert-compatibility profile, but are not yet available in *Connext Cert*

### 6.3.1 Compiling with compatibility for Connext Cert

To compile *Connext Micro* with the Cert-compatibility profile, you must use one of the available CERT architectures. To get a list of available CERT architectures, please use the following command:

```
cd <rti_me install directory>
resource/scripts/rtime-make --list
```

Architectures ending in CERT (e.g, x64Linux6gcc13.3.0CERT) are representative of Cert-compatibility profiles. To compile, use the following command:

```
cd <rti_me install directory>
resource/scripts/rtime-make --target x64Linux6gcc13.3.0CERT <other options>
```

The library is generated in the directory `lib/x64Linux6gcc13.3.0CERT`.

### 6.3.2 Compiling applications with compatibility for Connext Cert

To compile an application with compatibility for Connext Cert, the application must be compiled with the `RTI_CERT=1` preprocessor flag. This can be achieved with one of the following methods:

- If a CMakeLists.txt file generated with rtiddsgen is used, pass `-DRTIME_CERT=true` to either `rtime-make` or cmake.
- Pass `-DRTI_CERT=1` directly to the C preprocessor

With `rtime-make` on Linux or macOS:

```
resource/scripts/rtime-make -target x64Linux6gcc13.3.0CERT -DRTIME_CERT=true --src-dir .␣
↪ <other options>
```

With `rtime-make` On Windows:

```
resource/scripts/rtime-make --target x64Win64VS2015CERT -DRTIME_CERT_eq_true --src-dir .␣
↪ <other options>
```

Please refer to *Example Applications* for more information about generating examples.

## 6.4 GUID Generation

*RTI Connext Micro* uses Globally Unique Identifiers (GUIDs), also called Universally Unique Identifiers (UUIDs), to distinguish between different *DomainParticipants* in the same domain. If GUIDs are not unique, the middleware cannot tell *DomainParticipants* and endpoints apart, which can lead to conflicts (e.g., multiple nodes appearing as the same entity), discovery failures, dropped samples, or other undefined behavior.

*Connext Micro* will generate GUIDs for supported platforms. However, under certain conditions (such as multiple *DomainParticipants* with the exact same configuration), the default GUID generation may cause GUID collisions. For this reason, *Connext Micro* provides an interface that can be used to generate a GUID by taking advantage of unique platform-specific characteristics.

This section provides guidelines and examples for how to generate a unique GUID.

### 6.4.1 GUID format

The Object Management Group (OMG) specification, defines a GUID as a 16-byte unique identifier. With the leftmost byte as byte 0, the format of the GUID is:

- Bytes 0-1 are reserved for the vendor;
- Bytes 2-11 are generated;
- Bytes 12-15 are reserved for the entity ID.

This leaves exactly 10 bytes to provide uniqueness. To avoid collisions, these 10 bytes must account for both spatial uniqueness (different physical boards or nodes on the network) and temporal uniqueness (different processes or system restarts on the same board).

### 6.4.2 General guidelines

The following steps are general guidelines for implementing your own GUID generation. These are especially critical in environments without a POSIX layer, filesystem, or battery-backed Real-Time Clock (RTC), where the risk of GUID collision is high due to a typically-deterministic execution environment.

1. Use the Silicon Unique ID (UID). Most modern microcontroller units (MCUs) have a factory-burned ID. Use this as your "Machine ID" to maintain spatial uniqueness. Alternatively, you can use the IP address and MAC address in conjunction with a counter.
2. Implement a mandatory boot counter. Since some clocks reset to 0 on power-on, you *must* implement a persistent counter in non-volatile memory (NVM).

   *Workflow:* Read counter → Increment → Save to NVM → Use value in GUID.
3. If your system lacks a True Random Number Generator (TRNG), seed your Psuedo Random Number Generator (PRNG) by reading the least significant bits of a floating (unconnected) ADC pin (if available).

### 6.4.3 Testing

Before deploying, you should perform these two validation tests:

1. Generate a GUID, reboot the device, and generate another. The IDs must be different even if the system clock resets to the same epoch.
2. Flash two identical boards with the same firmware and power them on at the exact same time. Verify that their GUIDs are unique.

### 6.4.4 Example

The following code snippet is an example function to generate a GUID for STM32, which is a bare-metal system with no OS. It combines the hardware unique ID (built into the silicon) for spatial uniqueness and a Flash-based boot counter for temporal uniqueness.

```
/* Standard addresses for STM32 Unique Device ID (96-bit) */
#define STM32_UUID_ADDR 0x1FFF7A10

RTI_PRIVATE RTI_BOOL
OSAPI_SystemBareMetal_generate_uuid(struct OSAPI_SystemUUID *uuid_out)
{
    uint32_t boot_count = 0;
    uint32_t *hw_id = (uint32_t *)STM32_UUID_ADDR;

    /* 1. SPATIAL: Use the Silicon UID (First 6 bytes of the budget) */
    /* We XOR the 96-bit ID into 48 bits to capture all silicon variance */
    uint32_t part1 = hw_id[0] ^ hw_id[2];
    uint16_t part2 = (uint16_t)(hw_id[1] & 0xFFFF);

    OSAPI_Memory_copy(&uuid_out->value[2], &part1, 4);
    OSAPI_Memory_copy(&uuid_out->value[6], &part2, 2);

    /* 2. TEMPORAL: Persistent Boot Counter (Next 4 bytes of the budget) */
    /* Implementation-specific: Read from EEPROM or Flash User Config Sector */
    boot_count = NV_Memory_Read_Boot_Counter();
    boot_count++;
    NV_Memory_Write_Boot_Counter(boot_count);

    OSAPI_Memory_copy(&uuid_out->value[8], &boot_count, 4);

    /* 3. Optional: If a TRNG (True Random Number Gen) is available,
       XOR some random bits into the boot_count for extra safety */
    // uint32_t noise = HAL_RNG_GetRandomNumber(&hrng);
    // *((uint32_t*)&uuid_out->value[8]) ^= noise;

    return RTI_TRUE;
}
```

### 6.4.5 Implementing a custom GUID generation function

If the provided GUID generation does not meet your requirements, you can implement your own by following these steps:

1. Get the native interface:

```
RTI_PRIVATE struct OSAPI_SystemI OSAPI_System_fv_SysIntf;

// Get the current system interface
OSAPI_System_get_native_interface(&OSAPI_System_fv_SysIntf);
```

2. Implement your custom GUID generation function:

```
RTI_PRIVATE RTI_BOOL
MyApp_custom_generate_uuid(struct OSAPI_SystemUUID *uuid_out)
{
    /* Your custom function goes here */
}
```

3. Override the system interface:

```
// Replace the generate_uuid function pointer
OSAPI_System_fv_SysIntf.generate_uuid = MyApp_custom_generate_uuid;

// Set the modified interface
OSAPI_System_set_interface(&OSAPI_System_fv_SysIntf);
```

## 6.5 FreeRTOS Platforms

The following table shows the currently supported FreeRTOS platforms.

Table 6.5: Supported FreeRTOS Platforms

| OS | Version | CPU | Network Stack | Toolchain | Architecture PIL | Architecture PSL |
|---|---|---|---|---|---|---|
| FreeRTOS | 10.0.0 | ARMv7E-M | LWiP 2.0.0 | GCC 10.3.1 | armv7emleElfgcc10.3.1 armv7em-leElfgcc10.3.1CERT | armv7em-leElfgcc10.3.1-FreeRTOS10.0 armv7em-leElfgcc10.3.1CERT-FreeRTOS10.0 |

### 6.5.1 Port overview

RTI ported *Connext Micro* to run on the FreeRTOS operating system with the lwIP protocol stack. This section contains some additional information on the hardware and software used.

RTI used S32G-VNP-RDB2 as the reference hardware. This reference kit has a S32G274A processor with 3x Arm Cortex-M7 LS pairs and 8MB system RAM. For a full description, please refer to the microcontroller documentation here.

NXP provides a comprehensive software ecosystem for its S32G platform. For development on the M7 core, NXP offers the S32 Design Studio (S32DS)—a free, Eclipse-based integrated development environment (IDE) that supports the full S32G family. S32DS includes a GCC-based C/C++ compiler, GDB debugger integration, and tools for flashing and debugging the M7 core.

To simplify embedded software development, NXP provides the Real-Time Drivers (RTD) 4.0 package. RTD includes all necessary low-level drivers and middleware components to build applications for the M7 core on S32G platforms. It is fully compliant with AUTOSAR 4.x and integrates with FreeRTOS and other lightweight RTOS environments. RTI used the following versions of the different components:

- RTD version V4.0.0
- S32 Design Studio V3.6.0
- FreeRTOS Kernel V10.4.6
- lwIP version V2.0.0

### 6.5.2 How to configure lwIP and FreeRTOS

RTI ported *Connext Micro* to FreeRTOS using the following lwIP and FreeRTOS configurations. These can be tuned according to your needs by modifying the examples below. Details about how to configure these third-party components can also be found in the FreeRTOS documentation and lwIP documentation.

lwIP

```
#ifndef LWIP_OPTS_H
#define LWIP_OPTS_H

/**
* @page misra_violations MISRA-C:2012 violations
*
* @section [global]
* Violates MISRA 2012 Advisory Rule 2.5, Global macro not referenced.
* The global macro will be used in function call of the module.
*/

/* ---------- Application APIs ---------- */
#define NO_SYS                      0
#define LWIP_RAW                    1
#define LWIP_SOCKET                 (NO_SYS==0)
#define LWIP_NETCONN                (NO_SYS==0)

/* ---------- ARP options ---------- */
#define LWIP_ARP                    1
#define ARP_TABLE_SIZE              10
#define ARP_QUEUEING                1

/* ---------- IP options ---------- */
#define LWIP_IPV6                   0

/* ---------- DHCP options ---------- */
/* Define LWIP_DHCP to 1 if you want DHCP configuration of interfaces */
#define LWIP_DHCP                   0
/* 1 if you want to do an ARP check on the offered address (recommended) */
#define DHCP_DOES_ARP_CHECK         (LWIP_DHCP)
```

(continues on next page)

(continued from previous page)

```
/* ---------- AUTOIP options ---------- */
#define LWIP_AUTOIP                     0
#define LWIP_DHCP_AUTOIP_COOP           (LWIP_DHCP && LWIP_AUTOIP)
#define LWIP_DHCP_AUTOIP_COOP_TRIES     3

/* Define IP_FORWARD to 1 if you wish to have the ability to forward
   IP packets across network interfaces. If you are going to run lwIP
   on a device with only one network interface, define this to 0. */
#define IP_FORWARD                      0

/* IP reassembly and segmentation.These are orthogonal even
* if they both deal with IP fragments */
#define IP_REASSEMBLY                   1
#define IP_REASS_MAX_PBUFS              10
#define MEMP_NUM_REASSDATA              10
#define IP_FRAG                         1

/* ---------- IP options ---------- */
#define LWIP_IPV4                       1
#define ICMP_TTL                        255
#define LWIP_ICMP                       1
#define LWIP_IGMP                       1

/* ---------- TCP options ---------- */
#define LWIP_TCP                1
#define TCP_TTL                 255

/* Controls if TCP should queue segments that arrive out of
   order. Define to 0 if your device is low on memory. */
#define TCP_QUEUE_OOSEQ         0

/* TCP Maximum segment size. */
#define TCP_MSS                 1536

/* TCP sender buffer space (bytes). */
```

(continues on next page)

(continued from previous page)

```
#define TCP_SND_BUF             TCP_MSS * 4

/* TCP sender buffer space (pbufs). This must be at least = 2 *
   TCP_SND_BUF/TCP_MSS for things to work. */
#define TCP_SND_QUEUELEN       (4 * TCP_SND_BUF/TCP_MSS)

/* TCP writable space (bytes). This must be less than or equal
   to TCP_SND_BUF. It is the amount of space which must be
   available in the tcp snd_buf for select to return writable */
#define TCP_SNDLOWAT           (TCP_SND_BUF/2)

/* TCP receive window. */
#define TCP_WND                 4096

/* Maximum number of retransmissions of data segments. */
#define TCP_MAXRTX              2

/* Maximum number of retransmissions of SYN segments. */
#define TCP_SYNMAXRTX           2

/* ---------- UDP options ---------- */
#define LWIP_UDP                1
#define LWIP_UDPLITE            LWIP_UDP
#define UDP_TTL                 255

/* ---------- Memory options ---------- */
/* MEM_ALIGNMENT: should be set to the alignment of the CPU for which
   lwIP is compiled. 4 byte alignment -> define MEM_ALIGNMENT to 4, 2
   byte alignment -> define MEM_ALIGNMENT to 2. */
/* MSVC port: intel processors do not need 4-byte alignment,
   but are faster that way! */

/*!
 * @brief Custom memcpy function used to optimize and improve the lwip copy speed process.
 *
 * This function copies len characters from memory area src to memory area dest using 64 bit alignment.
```

(continues on next page)

(continued from previous page)

```
 */
void memcpy_64(uint64_t *dst, uint64_t * src, unsigned int len);

/*!
 * @brief Custom memcpy function used to optimize and improve the lwip copy speed process.
 *
 * This function copies len characters from memory area src to memory area dest.
 */
void memcpy_custom(void *dst, const void * src, unsigned int len);

#define MEMCPY(dst,src,len)          memcpy(dst,src,len)

/* Memory alignment must match your architecture - for ARM Cortex M7, 8-byte alignment is required */
#define MEM_ALIGNMENT                8

/* MEM_SIZE: the size of the heap memory. If the application will send
a lot of data that needs to be copied, this should be set high. */
#define MEM_SIZE                     128*1024

/* MEMP_NUM_PBUF: the number of memp struct pbufs. If the application
   sends a lot of data out of ROM (or other static memory), this
   should be set high. */
#define MEMP_NUM_PBUF                8

/* MEMP_NUM_RAW_PCB: the number of UDP protocol control blocks. One
   per active RAW connection. */
#define MEMP_NUM_RAW_PCB             4

/* MEMP_NUM_UDP_PCB: the number of UDP protocol control blocks. One
   per active UDP connection. */
#define MEMP_NUM_UDP_PCB             4

/* MEMP_NUM_TCP_PCB: the number of simultaneously active TCP connections. */
#define MEMP_NUM_TCP_PCB             4

/* MEMP_NUM_TCP_PCB_LISTEN: the number of listening TCP connections. */
```

(continues on next page)

(continued from previous page)

```
#define MEMP_NUM_TCP_PCB_LISTEN     4

/* MEMP_NUM_TCP_SEG: the number of simultaneously queued TCP segments.*/
#define MEMP_NUM_TCP_SEG            64

/* MEMP_NUM_SYS_TIMEOUT: the number of simultaneously active timeouts. */
#define MEMP_NUM_SYS_TIMEOUT        15

/* The following four are used only with the sequential API and can be
   set to 0 if the application only will use the raw API. */
/* MEMP_NUM_NETBUF: the number of struct netbufs. */
#define MEMP_NUM_NETBUF             16

/* MEMP_NUM_NETCONN: the number of struct netconns. */
#define MEMP_NUM_NETCONN            32

/* MEMP_NUM_TCPIP_MSG_*: the number of struct tcpip_msg, which is used
   for sequential API communication and incoming packets. Used in
   src/api/tcpip.c. */
#define MEMP_NUM_TCPIP_MSG_API      40  /* Increased from 20 */
#define MEMP_NUM_TCPIP_MSG_INPKT    30  /* Increased from 20 */

/* MEM_USE_POOLS==1: Use an alternative to malloc()
   To use this, MEMP_USE_CUSTOM_POOLS also has to be enabled. */
#define MEM_USE_POOLS               0
#define MEMP_USE_CUSTOM_POOLS       0

/* ---------- Pbuf options ---------- */
/* PBUF_POOL_SIZE: the number of buffers in the pbuf pool. */
#define PBUF_POOL_SIZE              32
#define PBUF_POOL_BUFSIZE           1536

/** SYS_LIGHTWEIGHT_PROT
* define SYS_LIGHTWEIGHT_PROT in lwipopts.h if you want inter-task protection
* for certain critical regions during buffer allocation, deallocation and memory
* allocation and deallocation. */
```

(continues on next page)

(continued from previous page)

```
#define SYS_LIGHTWEIGHT_PROT        (NO_SYS==0)

/* ---------- Statistics options ---------- */
#define LWIP_STATS                      1
#define LWIP_STATS_DISPLAY              1

#if LWIP_STATS
#define LINK_STATS                      1
#define IP_STATS                        1
#define ICMP_STATS                      1
#define IGMP_STATS                      1
#define IPFRAG_STATS                    1
#define UDP_STATS                       1
#define TCP_STATS                       1
#define MEM_STATS                       1
#define MEMP_STATS                      1
#define PBUF_STATS                      1
#define SYS_STATS                       1
#endif /* LWIP_STATS */

/* ---------- PPP options ---------- */
#define PPP_SUPPORT                     0

/* ---------- Other options ---------- */

/* (LWIP_ETHERNET == 1) Enable ETHERNET support even though ARP might be disabled */
#define LWIP_ETHERNET                   1

#define TCPIP_MBOX_SIZE                 24

#define DEFAULT_UDP_RECVMBOX_SIZE       1500
#define DEFAULT_TCP_RECVMBOX_SIZE       1500
#define DEFAULT_RAW_RECVMBOX_SIZE       1500
#define DEFAULT_ACCEPTMBOX_SIZE         1500
#define LWIP_NETIF_TX_SINGLE_PBUF       1
```

(continues on next page)

(continued from previous page)

```
#define LWIP_SUPPORT_CUSTOM_PBUF        1

#define LWIP_SNMP                       LWIP_UDP
#define MIB2_STATS                      LWIP_SNMP

#define LWIP_DNS                        LWIP_UDP
#define LWIP_MDNS_RESPONDER             LWIP_UDP

#define LWIP_NUM_NETIF_CLIENT_DATA      (LWIP_MDNS_RESPONDER)

#define LWIP_HAVE_LOOPIF                0
#define LWIP_NETIF_LOOPBACK             0

#define TCP_LISTEN_BACKLOG              1

#define LWIP_COMPAT_SOCKETS             1
#define LWIP_SO_RCVTIMEO                1
#define LWIP_SO_RCVBUF                  1

#define LWIP_TCPIP_CORE_LOCKING         1

#define LWIP_NETIF_LINK_CALLBACK        1
#define LWIP_NETIF_STATUS_CALLBACK      1

#define LWIP_NETCONN_SEM_PER_THREAD     0

#define LWIP_SOCKET_SET_ERRNO           1

#define LWIP_NETIF_HOSTNAME             1

/* ---------- TIMEVAL options ---------- */
#define LWIP_TIMEVAL_PRIVATE            0

/* ---------- Thread options ---------- */
#define TCPIP_THREAD_NAME               "tcpip_thread"
#define TCPIP_THREAD_STACKSIZE          1024*64
```

(continues on next page)

(continued from previous page)

```
#define TCPIP_THREAD_PRIO               4
#define DEFAULT_THREAD_STACKSIZE        4096
#define DEFAULT_THREAD_PRIO             3

/**
 * LWIP_SO_RCVBUF==1: Enable SO_RCVBUF processing.
 */
#define LWIP_SO_RCVBUF                  1

#define LWIP_SOCKET_SELECT              0

#endif /* LWIP_OPTS_H */
```

FreeRTOS

```
#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION                    1
#define configCPU_CLOCK_HZ                      ( 400000000UL )
#define configTICK_RATE_HZ                      ((TickType_t) 1000 )
#define configMAX_PRIORITIES                    10
#define configMINIMAL_STACK_SIZE                RTI_configMINIMAL_STACK_SIZE
#define configMAX_TASK_NAME_LEN                 16
```

(continues on next page)

(continued from previous page)

```
#define configUSE_16_BIT_TICKS                      0
#define configIDLE_SHOULD_YIELD                     1
#define configUSE_PREEMPTION                        1
#define configNUM_THREAD_LOCAL_STORAGE_POINTERS     2
#define configUSE_PORT_OPTIMISED_TASK_SELECTION     0
#define configUSE_TASK_NOTIFICATIONS                1
#define configUSE_TIME_SLICING                      0
#define configUSE_NEWLIB_REENTRANT                  0
#define configENABLE_BACKWARD_COMPATIBILITY         1
#define configUSE_POSIX_ERRNO                       0
#define configUSE_APPLICATION_TASK_TAG              0
#define configRECORD_STACK_HIGH_ADDRESS             1

/* Definition assert() function. */
#define configASSERT(x)                             if((x)==0) { taskDISABLE_INTERRUPTS(); __asm__ volatile ("bkpt #0");␣
↪for( ;; ); }

/* The highest interrupt priority that can be used by any interrupt service
routine that makes calls to interrupt safe FreeRTOS API functions.  DO NOT CALL
INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER
PRIORITY THAN THIS! (higher priorities are lower numeric values. */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY ( 2 )

/* Memory allocation related definitions. */
#define configSUPPORT_STATIC_ALLOCATION             0
#define configSUPPORT_DYNAMIC_ALLOCATION            1
#define configTOTAL_HEAP_SIZE                       (( size_t ) (__heap_size__) )
#define configAPPLICATION_ALLOCATED_HEAP            4

/* Hook function related definitions. */
#define configUSE_IDLE_HOOK                         0
#define configUSE_TICK_HOOK                         0
#define configUSE_MALLOC_FAILED_HOOK                1
#define configCHECK_FOR_STACK_OVERFLOW              1
#define configUSE_DAEMON_TASK_STARTUP_HOOK          0
```

(continues on next page)

(continued from previous page)

```
/* Run time and task stats gathering related definitions. */
#define configGENERATE_RUN_TIME_STATS                1
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()
#define portGET_RUN_TIME_COUNTER_VALUE()             xTaskGetTickCount()
#define configUSE_TRACE_FACILITY                     1
#define configUSE_STATS_FORMATTING_FUNCTIONS         1

/* Co-routine related definitions. */
#define configUSE_CO_ROUTINES                        0
#define configMAX_CO_ROUTINE_PRIORITIES              2

/* SEMAPHORES and MUTEXS */
#define configUSE_MUTEXES                            1
#define configUSE_RECURSIVE_MUTEXES                  1
#define configUSE_COUNTING_SEMAPHORES                1

/* Software timer related definitions. */
/* Software timer related definitions. */
#define configUSE_TIMERS                             1
#define configTIMER_TASK_PRIORITY                    6
#define configTIMER_QUEUE_LENGTH                     16
#define configTIMER_TASK_STACK_DEPTH                 (1024*2)

/* Tickless Idle Mode */
#define configUSE_TICKLESS_IDLE                      0

/* QUEUE */
#define configQUEUE_REGISTRY_SIZE                    8
#define configUSE_QUEUE_SETS                         0

/* Optional functions - most linkers will remove unused functions anyway. */
#define INCLUDE_vTaskPrioritySet                     1
#define INCLUDE_uxTaskPriorityGet                    1
#define INCLUDE_vTaskDelete                          1
#define INCLUDE_vTaskSuspend                         1
#define INCLUDE_vTaskDelayUntil                      1
```

(continues on next page)

(continued from previous page)

```
#define INCLUDE_vTaskDelay                          1
#define INCLUDE_xTaskGetSchedulerState              1
#define INCLUDE_xTaskGetCurrentTaskHandle           1
#define INCLUDE_uxTaskGetStackHighWaterMark         1
#define INCLUDE_uxTaskGetStackHighWaterMark2        0
#define INCLUDE_xTaskGetIdleTaskHandle              0
#define INCLUDE_eTaskGetState                       1
#define INCLUDE_xEventGroupSetBitFromISR            1
#define INCLUDE_xTimerPendFunctionCall              1
#define INCLUDE_xTaskAbortDelay                     1
#define INCLUDE_xTaskGetHandle                      1
#define INCLUDE_xTaskResumeFromISR                  1
#define INCLUDE_xQueueGetMutexHolder                1

/* Run time stats gathering definitions. */
#ifdef __GNUC__
    /* The #ifdef just prevents this C specific syntax from being included in
    assembly files. */
    void vMainConfigureTimerForRunTimeStats( void );
    unsigned long ulMainGetRunTimeCounterValue( void );
#endif

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
  #define configPRIO_BITS                           __NVIC_PRIO_BITS
#else
  #define configPRIO_BITS                           4
#endif

/* The lowest interrupt priority that can be used in a call to a "set priority"
function. */
#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY  (0x0F)

/* Interrupt priorities used by the kernel port layer itself.  These are generic
to all Cortex-M ports, and do not rely on any particular library functions. */
#ifndef configKERNEL_INTERRUPT_PRIORITY
```

(continues on next page)

(continued from previous page)

```
  #define configKERNEL_INTERRUPT_PRIORITY         (configLIBRARY_LOWEST_INTERRUPT_PRIORITY << (8-configPRIO_BITS))
#endif

/* !!!! configMAX_SYSCALL_INTERRUPT_PRIORITY must not be set to zero !!!!
See http://www.FreeRTOS.org/RTOS-Cortex-M3-M4.html. */
#ifndef configMAX_SYSCALL_INTERRUPT_PRIORITY
  #define configMAX_SYSCALL_INTERRUPT_PRIORITY    (configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY << (8-configPRIO_BITS))
#endif

/* Definitions that map the FreeRTOS port interrupt handlers to their CMSIS
standard names. */
#define vPortSVCHandler                           SVC_Handler
#define xPortPendSVHandler                        PendSV_Handler
#define xPortSysTickHandler                       SysTick_Handler


#endif /* FREERTOS_CONFIG_H */
```

### 6.5.3 How the PIL was built for FreeRTOS

This section describes how RTI built the Platform Independent Library (PIL) for FreeRTOS.

The following table shows the compiler flags RTI used to create the PIL for FreeRTOS platforms:

Table 6.6: PIL Compiler Flags for FreeRTOS Platforms

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv7emleElfgcc10.3.1 | Static Release | –specs=nosys.specs –specs=nano.specs -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_stubs.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1" -DRTI_PIL=1 -D___freertos___ -std=c99 -nostdinc -fstrict-aliasing -fsigned-char -O -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums -DNDEBUG |
| | Static Debug | –specs=nosys.specs –specs=nano.specs -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_stubs.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1" -DRTI_PIL=1 -D___freertos___ -std=c99 -nostdinc -fstrict-aliasing -fsigned-char -O0 -g -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums |
| armv7emleElfgcc10.3.1CERT | Static Release | –specs=nosys.specs –specs=nano.specs -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_stubs.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1CERT" -DRTI_CERT -DRTI_PIL=1 -D___freertos___ -std=c99 -nostdinc -fstrict-aliasing -fsigned-char -O -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums -DNDEBUG |

continues on next page

Table 6.6 – continued from previous page

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Static Debug | -specs=nosys.specs –specs=nano.specs -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_stubs.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1CERT" -DRTI_CERT -DRTI_PIL=1 -D___freertos___ -std=c99 -nostdinc -fstrict-aliasing -fsigned-char -O0 -g -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums |

### 6.5.4 Building the PSL from source for FreeRTOS platforms

This section describes how to build your own PSL for FreeRTOS.

*Connext Micro* includes support to compile Platform Support Libraries (PSL) for FreeRTOS using CMake. Refer to *Setting up the build environment* before continuing with the following instructions.

1. Make sure CMake is in the path.
2. Define the following environment variables:
   - `CONFIG_PATH`: Path to where the `FreeRTOSConfig.h` and `lwipopts.h` files are located.
   - `FREERTOS_PATH`: Path to the FreeRTOS source code and header files.
   - `LWIP_PATH`: Path to the lwIP source code and header files.
   - `LWIP_PORTS_PATH`: Path to the lwIP ports folder that contains cc.h.
   - `RTD_PATH`: Path to the folder that contains the eclipse plugins of RTD.
   - `PATH`: Update your path with the location of the C and C++ compiler. By default, `arm-none-eabi-gcc` and `arm-none-eabi-g++` are used as C and C++ compilers.
3. Enter the following command:

```
RTIMEHOME/resource/scripts/rtime-make --target armv7emleElfgcc10.3.1-FreeRTOS10.0 \
                      -G "Unix Makefiles" --build
```

**Note:** `rtime-make` uses the architecture specified with `--target` to determine a few settings needed by *Connext Micro*. Please refer to *Preparing to build* for details.

4. The *Connext Micro* PSL is available in:

```
RTIMEHOME/lib/armv7emleElfgcc10.3.1-FreeRTOS10.0
```

### 6.5.5 Building FreeRTOS applications with Connext Micro

This section describes how RTI built the Platform Support Library (PSL) for FreeRTOS platforms. You must build applications with compatible flags to the PIL and PSL in order to operate with *Connext Micro*. The PSL must also be binary compatible with the PIL. Applications must not specify the `RTI_PSL` or `RTI_PIL` preprocessor definitions.

The following table shows the compiler flags and required options that RTI used to build the PSL for FreeRTOS platforms. When you build the PSL with `rtime-make`, the `--target` argument automatically adds all the necessary flags for the specified architecture.

Table 6.7: PSL Compiler Flags for FreeRTOS Platforms

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv7emleElfgcc10.3.1-FreeRTOS10.0 | Static Release | –specs=nosys.specs –specs=nano.specs -DNDEBUG -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_freertos.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1-FreeRTOS10.0" -DRTI_PSL=1 -DUSING_RTD=1 -D_TIMEVAL_DEFINED=0 -D___freertos___ -std=c99 -fstrict-aliasing -fsigned-char -O -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums -DNDEBUG |

continues on next page

Table 6.7 – continued from previous page

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Static Debug | –specs=nosys.specs –specs=nano.specs -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_freertos.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1-FreeRTOS10.0" -DRTI_PSL=1 -DUSING_RTD=1 -D_TIMEVAL_DEFINED=0 -D__freertos__ -std=c99 -fstrict-aliasing -fsigned-char -O0 -g -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums |
| armv7em-leElfgcc10.3.1CERT-FreeRTOS10.0 | Static Release | –specs=nosys.specs –specs=nano.specs -DNDEBUG -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_freertos.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1CERT-FreeRTOS10.0" -DRTI_CERT -DRTI_PSL=1 -DUSING_RTD=1 -D_TIMEVAL_DEFINED=0 -D__freertos__ -std=c99 -fstrict-aliasing -fsigned-char -O -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums -DNDEBUG |
| | Static Debug | –specs=nosys.specs –specs=nano.specs -DOSAPI_CC_DEF_H=osapi/osapi_cc_gcc.h -DOSPSL_OS_DEF_H=rti_me_psl/ospsl/ospsl_os_freertos.h -DRTIME_TARGET_NAME="armv7emleElfgcc10.3.1CERT-FreeRTOS10.0" -DRTI_CERT -DRTI_PSL=1 -DUSING_RTD=1 -D_TIMEVAL_DEFINED=0 -D__freertos__ -std=c99 -fstrict-aliasing -fsigned-char -O0 -g -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -fno-short-enums |

### 6.5.6 System tick rollovers

The OMG standard does not specify how an implementation of DDS should handle counter rollover. By default, *Connext Micro* does not check for rollover of the system tick count on FreeRTOS platforms.

The OSAPI_SystemFreeRTOS_get_time function uses the system tick count to calculate the time in milliseconds since the system started. It does so by multiplying the number of ticks since the system started by the FreeRTOS-defined constant `portTICK_RATE_MS`. *Connext Micro* stores the result in an unsigned 32-bit variable.

Two conditions can result from this calculation: the tick count can rollover, and the resulting calculation can be greater than the size of an `unsigned int` (2^32). This can cause an overflow, which would result in time appearing to go backwards.

If you need to change or mitigate this behavior, you can alter the source code for the reference implementation of OSAPI_SystemFreeRTOS_get_time (or provide your own implementation) and rebuild the PSL.

## 6.6 Linux Platforms

The following table shows the currently supported Linux platforms.

Table 6.8: Supported Linux Platforms

| OS | Version | CPU | Network Stack | Toolchain | Architecture PIL | Architecture PSL |
|---|---|---|---|---|---|---|
| Ubuntu | 24.04 LTS | x64 | OS Default | GCC 13.3.0 | x86_64leElfgcc13.3.0<br>x86_64leElfgcc13.3.0CERT | x86_64leElfgcc13.3.0-Linux6<br>x86_64leElfgcc13.3.0CERT-Linux6 |
| Ubuntu | 24.04 LTS | ARMv8 (64-bit) | OS Default | GCC 13.3.0 | armv8leElfgcc13.3.0<br>armv8leElfgcc13.3.0CERT | armv8leElfgcc13.3.0-Linux6<br>armv8leElfgcc13.3.0CERT-Linux6 |
| Yocto | 5.15.96 | ARMv8-A (64-bit) | OS Default | GCC 11.3.1 | armv8aleElfgcc11.3.1<br>armv8aleElfgcc11.3.1CERT | armv8aleElfgcc11.3.1-Linux5<br>armv8aleElfgcc11.3.1CERT-Linux5 |

### 6.6.1 How the PIL was built for Linux platforms

This section describes how RTI built the Platform Independent Library (PIL) for Linux.

The following table shows the compiler flags RTI used to create the PIL for Linux platforms:

Table 6.9: PIL Compiler Flags for Linux Platforms

| Architecture | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| x86_64leElfgcc13.3.0 | Static Release | **C Flags:** -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG **C++ Flags:** -std=c++11 -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG |
| | Shared Release | **C Flags:** -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG **C++ Flags:** -std=c++11 -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC **C++ Flags:** -std=c++11 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |

continues on next page

Table 6.9 – continued from previous page

| Architecture | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Shared Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC **C++ Flags:** -std=c++11 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |
| x86_64leElfgcc13.3.0CERT | Static Release | **C Flags:** -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |
| armv8leElfgcc13.3.0 | Static Release | **C Flags:** -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG **C++ Flags:** -std=c++11 -fsigned-char -nostdinc -O2 -fstrict-aliasing -fPIC -DNDEBUG |
| | Shared Release | **C Flags:** -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG **C++ Flags:** -std=c++11 -fsigned-char -nostdinc -O2 -fstrict-aliasing -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC **C++ Flags:** -std=c++11 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |
| | Shared Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC **C++ Flags:** -std=c++11 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |
| armv8leElfgcc13.3.0CERT | Static Release | **C Flags:** -fsigned-char -O2 -nostdinc -fstrict-aliasing -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |
| armv8aleElfgcc11.3.1 | Static Release | **C Flags:** -nostdinc -fstrict-aliasing -fsigned-char -O -DNDEBUG -fPIC -DNDEBUG **C++ Flags:** -O -DNDEBUG -fPIC |
| | Shared Release | **C Flags:** -nostdinc -fstrict-aliasing -fsigned-char -O -DNDEBUG -fPIC -DNDEBUG **C++ Flags:** -O -DNDEBUG -fPIC |
| | Static Debug | **C Flags:** -std=c99 -nostdinc -fstrict-aliasing -fsigned-char -O0 -g -fPIC **C++ Flags:** -O0 -g -fPIC |
| | Shared Debug | **C Flags:** -std=c99 -nostdinc -fstrict-aliasing -fsigned-char -O0 -g -fPIC **C++ Flags:** -O0 -g -fPIC |
| armv8aleElfgcc11.3.1CERT | Static Release | **C Flags:** -nostdinc -fstrict-aliasing -fsigned-char -O -DNDEBUG |
| | Static Debug | **C Flags:** -fsigned-char -nostdinc -fstrict-aliasing -O0 -g |

**Warning:** The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, *Connext Micro* makes one direct call to the C library API `qsort`. In addition, GCC may insert direct calls to GLIBC functions and other required functions, such as default C++ constructors and destructors. For this reason, it is necessary to use a GCC version that is compatible with the GCC version used to build the platform independent libraries, or provide a C library with an implementation of the required functions. Future versions of *RTI Connext Micro* will remove these dependencies.

### 6.6.2 Building the PSL from source for Linux platforms

Refer to *Building the PSL* for instructions on how to build your own Platform Support Library (PSL) for Linux platforms.

The following table shows the compiler flags and required options that RTI used to build the PSL for Linux platforms. When you build the PSL with `rtime-make`, the `--target` argument automatically adds all the necessary flags for the specified architecture.

Table 6.10: PSL Compiler Flags for Linux Platforms

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| x86_64leElfgcc13.3.0-Linux6 | Static Release | **C Flags:** -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG **C++ Flags:** -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG |
| | Shared Release | **C Flags:** -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG **C++ Flags:** -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -g -fPIC **C++ Flags:** -fstrict-aliasing -fsigned-char -g -fPIC |
| | Shared Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -g -fPIC **C++ Flags:** -fstrict-aliasing -fsigned-char -g -fPIC |
| x86_64leElfgcc13.3.0CERT-Linux6 | Static Release | **C Flags:** -std=c99 -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -g -fPIC |
| armv8leElfgcc13.3.0-Linux6 | Static Release | **C Flags:** -std=c99 -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG **C++ Flags:** -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG |

continues on next page

Table 6.10 – continued from previous page

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Shared Release | **C Flags:** -std=c99 -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG **C++ Flags:** -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -g -fPIC **C++ Flags:** -fstrict-aliasing -fsigned-char -g -fPIC |
| | Shared Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -g -fPIC **C++ Flags:** -fstrict-aliasing -fsigned-char -g -fPIC |
| armv8leElfgcc13.3.0CERT-Linux6 | Static Release | **C Flags:** -std=c99 -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -g -fPIC |
| armv8aleElfgcc11.3.1-Linux5 | Static Release | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -O -DNDEBUG **C++ Flags:** -O -fPIC -DNDEBUG |
| | Shared Release | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -O -DNDEBUG **C++ Flags:** -O -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -O0 -g -fPIC **C++ Flags:** -O0 -g |
| | Shared Debug | **C Flags:**-std=c99 -fstrict-aliasing -fsigned-char -O0 -g -fPIC **C++ Flags:** -O0 -g |
| armv8aleElfgcc11.3.1CERT-Linux5 | Static Release | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -O -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fstrict-aliasing -fsigned-char -O0 -g |

## 6.7 MicroSAR Platforms

The following table shows the currently supported MicroSAR platforms.

Table 6.11: Supported MicroSAR Platforms

| OS | Version | CPU | Network Stack | Toolchain | Architecture PIL | Architecture PSL |
|---|---|---|---|---|---|---|
| MicroSAR | 29.06.20 (AUTOSAR 4.2.2) | TC397 | N/A | Tasking 6.3r1p7 | tc39xtElfTasking6.3r1 tc39xtElfTasking6.3r1CERT | tc39xtElfTasking6.3r1-MICROSAR4 tc39xtElfTasking6.3r1CERT-MICROSAR4 |

### 6.7.1 Port overview

*Connext Micro* includes support for AUTOSAR™ and enables *Connext Micro* applications to run on AUTOSAR systems. The support has been tested on the Infineon™ AURIX™ Application Kit TC397 TFT development board with Vector™ AUTOSAR implementation version 4.2.2 and compiler TASKING™ v6.3r1.

This manual explains how to compile and configure *Connext Micro* to run on AUTOSAR systems and the AUTOSAR configuration required by *Connext Micro*.

### 6.7.2 How to configure MicroSAR

#### DDS CDD configuration

---

**Note:** See *Generating AUTOSAR Applications* for instructions on how to generate an AUTOSAR CDD with Micro Application Generator.

---

*Connext Micro* is integrated as an AUTOSAR Complex Device Driver (CDD) SW-Component with the short name `DdsCddType`. The CDD does not support multiple instantiation. The following runnables are required in the CDD:

Table 6.12: Required Runnables

| Runnable | Trigger Type | Period/Condition | Concurrent | Required | Purpose |
|---|---|---|---|---|---|
| Start | INIT-EVENT | Once at ECU startup | No | Always | Initialization runnable. Called once when the RTE starts. This runnable initializes the DDS CDD. |
| Run | TIMING-EVENT | Periodic | No | Always | Periodic maintenance runnable. Maintains DDS QoS protocols (Discovery, Liveliness, Reliability), and processes internal timers. This is the main "heartbeat" of the DDS middleware on AUTOSAR. |
| RxIndication | Called by TcpIp stack | On UDP packet reception | No | Only if `use_udp_thread` is TRUE[1] | Called by the AUTOSAR TcpIp stack (via socket owner or PDU callout) when a UDP packet is received. When `use_udp_thread` is TRUE, this runnable copies the received data into an internal buffer and fires the Internal Triggering Point (ITP_DdsCdd_RxIndication) to wake up the ProcessData runnable.<br>The TcpIp stack calls one of the following *Connext Micro* callbacks depending on the socket owner configuration:<br>• `NETIO_Autosar_TcpIp_udp_rx_indication()` — when a socket owner is configured<br>• `NETIO_Autosar_TcpIp_pdu_callout()` — when no socket owner is configured |
| ProcessData | INTERNAL-TRIGGER-OCCURRED-EVENT | Triggered by ITP from RxIndication | No | Only if `use_udp_thread` is TRUE | Processes buffered UDP packets that were stored by RxIndication. It iterates through the internal receive buffer and dispatches each packet to the appropriate DDS DataReader via `NETIO_Autosar_udp_receive_callback()`. |

[1] When `use_udp_thread` is FALSE, incoming UDP packets are processed directly in the TcpIp callback context (synchronously), and the RxIndication/ProcessData runnables are not needed.

### BSW configuration

### OS configuration

The DDS CDD requires the following OS resources:

- STANDARD OS Resource: The DDS CDD requires one OS resource for internal synchronization (mutexes/semaphores). The resource ID used is determined by the `mutex_resource_id` field in `OSAPI_PortProperty`.
- Events: If semaphores are used (`semaphore_max_count > 0`), AUTOSAR events must be configured. The events are defined by `first_give_event` and `first_timeout_event` in `OSAPI_PortProperty`. Each event ID must be a power of 2.
- Alarms: If semaphores are used, AUTOSAR alarms are required for timeout support, starting from `first_alarm`.
- Task: At least one dedicated OS task should be configured for DDS CDD runnables. The task priority should be higher than application SW-Cs but lower than critical BSW tasks (Ethernet driver, TcpIp stack).

  Stack size recommendations depend on which runnables are mapped to each task:

  – **DdsCddRun runnable:** Allocate at least 2 KB of stack space for the task where this runnable is mapped.

  – **RxIndication and ProcessData runnables** (when `use_udp_thread` is TRUE): Allocate at least 5 KB of stack space for that task. Both runnables should be mapped to the same task.

  – **Application-defined read/write runnables:** The integrator is responsible for creating runnables that read from DDS *DataReaders* and write to DDS *DataWriters* based on their application's topics. These runnables call *Connext Micro* APIs (e.g., `DataReader_take()`, `DataWriter_write()`). For the task(s) where these runnables are mapped, allocate at least 8 KB of additional stack space for DDS operations.

---

**Note:** If all runnables are mapped to a single task, allocate at least 15 KB total.

---

- Spinlocks (optional): If `sync_type` is set to `OSAPI_AUTOSAR_SYNCKIND_SPINLOCK`, a spinlock must be configured in the OS and its ID set in `spinlock_id`.

### TcpIp configuration

The DDS CDD interfaces with the AUTOSAR TcpIp BSW module for all network communication. The following configuration is required:

- Socket Owner: Configure a socket owner for the DDS CDD so that *Connext Micro* can create and manage its own UDP sockets. Register the following callbacks:
  - RxIndication: `NETIO_Autosar_TcpIp_udp_rx_indication` (called when UDP data is received)
  - LocalIpAddrAssignmentChg: In this callback, DDS CDD can begin its initialization.
- UDP Sockets: DDS requires at minimum 2 UDP sockets in unicast mode and 3 UDP sockets if multicast is enabled (2 unicast + 1 multicast). Ensure the TcpIp module is configured to support this number of sockets.
- IP Address: A static or DHCP-assigned IP address must be available. The DDS CDD waits for the IP address to be assigned before proceeding with full initialization.

### BswM configuration

DDS entities (in particular the *DomainParticipant*) must only be created once the TcpIp stack is up and running with an IP address assigned, because entity creation triggers discovery messages that are sent to other peers on the network.

We recommend the following approach:

1. The TcpIp stack initializes and assigns an IP address.
2. The LocalIpAddrAssignmentChg callback for the socket owner fires, indicating TcpIp is ready.
3. Only after this callback has been received should the DDS CDD proceed with creating DDS entities.

---

**Note:** The exact mechanism to ensure TcpIp is ready before DDS entity creation is configurable and may vary per project. The key requirement is that DDS entities are not created until an IP address is available. One common approach is to use the BswM to gate `Rte_Start()` until after LocalIpAddrAssignmentChg has been received. Alternatively, the DDS CDD state machine can handle this internally by deferring entity creation until the callback sets a readiness flag.

---

### DomainParticipant configuration

When creating a *DomainParticipant*, the `participant_id` field in `DDS_DomainParticipantQos.protocol` must be set to a unique value for each participant in the same domain on the same node. This ID is used by the RTPS protocol to distinguish participants and must not conflict with other participants on the network.

```
struct DDS_DomainParticipantQos dp_qos = DDS_DomainParticipantQos_INITIALIZER;

/* Set participant ID - must be unique per domain */
dp_qos.protocol.participant_id = 0;

participant = DDS_DomainParticipantFactory_create_participant(
    dpf, domain_id, &dp_qos, NULL, DDS_STATUS_MASK_NONE);
```

---

**Note:** If `participant_id` is not set explicitly, the default value depends on the platform. On AUTOSAR platforms, always set this field explicitly to ensure deterministic behavior and avoid conflicts with other participants.

---

### OSAPI_PortProperty reference

The `OSAPI_PortProperty` structure contains all platform-specific configuration for the *Connext Micro* AUTOSAR port. It is set via `OSPSL_AutosarSystem_set_property()` before `OSAPI_System_initialize()` is called. The default initializer is `OSAPI_PortProperty_INITIALIZER`. All fields default to `0`, `NULL`, or `FALSE`, except `enable_thread_safe_heap` which defaults to `TRUE` and `sync_type` which defaults to `OSAPI_AUTOSAR_SYNCKIND_RESOURCES`.

### System configuration

Table 6.13: System configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `timer_resolution_ms` | `uint32` | 0 | Timer resolution in milliseconds. Determines the granularity of the DDS system clock. The DDS Run runnable period should match this value. A typical value is 10 (matching the 10 ms Run period). |

### Heap configuration

*Connext Micro* on AUTOSAR uses a static heap (no dynamic malloc/free). Memory must be pre-allocated as one or more heap areas and provided to the port via these properties.

Table 6.14: Heap Configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `number_of_heap_areas` | uint32 | 0 | Number of static memory areas provided for the DDS heap. Must be at least 1. |
| `heap_area_size` | const uint32* | NULL | Pointer to an array of sizes (in bytes) for each heap area. The first area must be large enough to hold at least the offset table for all memory areas (`sizeof(uint32) * (number_of_heap_areas + 1)` for alignment). |
| `heap_area` | const char** | NULL | Pointer to an array of pointers to the pre-allocated memory buffers. Each buffer must be non-NULL and its corresponding size must be non-zero. |
| `enable_thread_safe_heap` | boolean | TRUE | Set to TRUE to enable thread-safe heap operations using the configured synchronization mechanism. Set to FALSE only in single-threaded configurations. |

**Note:** The AUTOSAR heap implementation does not support freeing memory. Once memory is allocated it cannot be returned to the heap. Plan your heap size accordingly.

### Mutex configuration

Table 6.15: Mutex Configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `sync_type` | OSAPI_Autosar_SyncKind_T | OSAPI_AUTOSAR_SYNCKIND_RESOURCES | Synchronization mechanism. Use OSAPI_AUTOSAR_SYNCKIND_RESOURCES to use AUTOSAR STANDARD resources (GetResource/ReleaseResource). Use OSAPI_AUTOSAR_SYNCKIND_SPINLOCK to use spinlocks (GetSpinlock/ReleaseSpinlock). Spinlock support is only available when RTIME_AUTOSAR_SPINLOCK_ENABLED is 1. |
| `mutex_resource_id` | ResourceType | 0 | The AUTOSAR resource ID allocated for DDS mutexes when sync_type is OSAPI_AUTOSAR_SYNCKIND_RESOURCES. |
| `spinlock_id` | uint16 | 0 | The AUTOSAR spinlock ID to use when sync_type is OSAPI_AUTOSAR_SYNCKIND_SPINLOCK. Only available when RTIME_AUTOSAR_SPINLOCK_ENABLED is 1. |

### Semaphore configuration

AUTOSAR does not provide native semaphore support. *Connext Micro* implements semaphores using AUTOSAR Events and Alarms. If your application uses DDS features that require semaphores (e.g., blocking reads), these fields must be configured.

Table 6.16: Semaphore Configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `semaphore_max_count` | uint32 | 0 | Maximum number of concurrent semaphores. Set to 0 to disable semaphore support. If non-zero, the `first_give_event`, `first_timeout_event`, and `first_alarm` fields must also be configured. The number of bits between the bit indices of `first_give_event` and `first_timeout_event` must be `semaphore_max_count`. |
| `first_give_event` | EventMaskType | 0 | The first AUTOSAR event mask used for semaphore "give" signals. Must be a power of 2 and non-zero when `semaphore_max_count > 0`. |
| `first_timeout_event` | EventMaskType | 0 | The first AUTOSAR event mask used for semaphore timeout signals. Must be a power of 2, non-zero, and different from `first_give_event` when `semaphore_max_count > 0`. |
| `first_alarm` | AlarmType | 0 | The first AUTOSAR alarm ID used for semaphore timeouts. Must be non-zero when `semaphore_max_count > 0`. The range [`first_alarm`, `first_alarm + semaphore_max_count`) must be valid alarm IDs. |

**Socket configuration**

Table 6.17: Socket Configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `use_socket_owner` | boolean | FALSE | Set to TRUE to use the AUTOSAR TcpIp Socket Owner mechanism for socket management. When TRUE, `get_socket` must point to a valid function. When FALSE, `send_data` must point to a valid function. Note: `use_socket_owner = FALSE` has not been fully tested (see Known Issues). |
| `max_receive_sockets` | uint8 | 0 | Maximum number of receive sockets that DDS will create. Must be at least 1. A typical value for DDS with unicast is 4; with multicast enabled, use 5. |
| `get_socket` | NETIO_Autosar_TcpIp_get_socket | NULL_PTR | Function pointer for obtaining a TcpIp socket. Required when `use_socket_owner` is TRUE. The function signature is: `Std_ReturnType (*)(TcpIp_DomainType domain, TcpIp_ProtocolType protocol, TcpIp_SocketIdType *socket)`. |
| `send_data` | NETIO_Autosar_TcpIp_send_dyn_data | NULL_PTR | Function pointer for sending UDP data. Required when `use_socket_owner` is FALSE. The function signature is: `Std_ReturnType (*)(const void *data_ptr, uint16 length, const TcpIp_SockAddrType *remote_addr_ptr)`. |
| `max_local_addr_id` | TcpIp_LocalAddrIdType | 0 | Maximum configured IP address identifier. Represents the range top for configured IP addresses [0..max_local_addr_id]. |
| `send_local_addr_id` | TcpIp_LocalAddrIdType | 0 | Local address ID to use for outgo- |

### UDP thread configuration

These fields control how incoming UDP packets are processed. When `use_udp_thread` is TRUE, received packets are buffered and processed asynchronously by the ProcessData runnable. When FALSE, packets are processed directly in the TcpIp callback context.

Table 6.18: UDP Thread Configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `use_udp_thread` | boolean | FALSE | Set to TRUE to enable asynchronous UDP packet processing via a dedicated task/runnable. When TRUE, the RxIndication and ProcessData runnables must be configured in the CDD. When FALSE, packets are processed synchronously in the TcpIp callback. |
| `udp_receive_task_id` | TaskType | 0 | DEPRECATED. TO BE REMOVED |
| `udp_packet_received_event` | EventMaskType | 0 | DEPRECATED. TO BE REMOVED |
| `number_of_rcv_buffers` | uint8 | 0 | Number of internal receive buffers for storing incoming UDP packets. Required when `use_udp_thread` is TRUE. Must be non-zero. |
| `rcv_buffer_size` | uint32 | 0 | Size in bytes of each receive buffer. Must accommodate the largest expected UDP packet. A typical value is 1500 (standard Ethernet MTU). Required when `use_udp_thread` is TRUE. Must be non-zero. |

### Logging configuration

These fields are only available when `OSAPI_ENABLE_LOG` is set to 1 at compile time.

Table 6.19: Logging Configuration

| Field | Type | Default | Description |
|---|---|---|---|
| `log_dst_address` | uint32 | 0 | Destination IP address for remote log output. Format is host-byte-order (e.g., 0xCA00A8C0 = 192.168.0.202). Set to 0 to disable remote logging. |
| `log_dst_port` | uint16 | 0 | Destination UDP port for remote log output. |

### Routing DDS errors to AUTOSAR DET

By default, *Connext Micro* does not call `Det_ReportError`. The AUTOSAR port ships a built-in display handler, `OSAPI_AutosarLog_default_display`, that routes every DDS log entry to the AUTOSAR Default Error Tracer (DET) via `Det_ReportError`. This handler is only available when `OSAPI_ENABLE_LOG` is set to 1 at compile time.

#### How the handler works

`OSAPI_AutosarLog_default_display` performs the following steps for each log entry:

1. Reads the current verbosity level via `OSAPI_Log_get_verbosity()`.
2. Checks the log entry kind (`OSAPI_LOGKIND_INFO`, `OSAPI_LOGKIND_WARNING`, `OSAPI_LOGKIND_ERROR`, `OSAPI_LOGKIND_PRECONDITION`) against the current verbosity. Entries below the configured verbosity threshold are silently dropped and do not produce a DET report.
3. Extracts the DDS `module_id` and `error_id` from the log entry's error code.
4. Packs both values into the two 8-bit DET fields (`ApiId` and `ErrorId`) as follows, to fit within DET's field-width constraints:
   - `ApiId[7:4]` = high nibble of `error_id` (bits [11:8])
   - `ApiId[3:0]` = `module_id[3:0]`
   - `ErrorId[7:0]` = low byte of `error_id` (bits [7:0])
5. Calls `Det_ReportError` with:
   - `ModuleId` = `RTIME_DDS_MODULE_ID` (`0x010A`, the RTI OMG Vendor ID)
   - `InstanceId` = `RTIME_DDS_INSTANCE_ID` (`0`)

- `ApiId` and `ErrorId` as packed above.

**How to enable the handler**

Call `OSAPI_Log_set_display_handler` during DDS initialization to replace the default display handler with the DET-reporting one:

```
#if OSAPI_ENABLE_LOG
OSAPI_Log_set_display_handler(
    (OSAPI_LogDisplay_T)&OSAPI_AutosarLog_default_display,
    NULL);
#endif
```

The second argument (`param`) is not used by `OSAPI_AutosarLog_default_display` and must be `NULL`.

---

**Note:** This handler respects the verbosity level configured via `OSAPI_Log_set_verbosity()`. Only log entries at or above the configured verbosity level will trigger a DET report. Guard the call with `#if OSAPI_ENABLE_LOG` to ensure it is only compiled when logging is enabled.

---

### Binding receive sockets to a specific interface

By default, *Connext Micro* binds receive sockets to `INADDR_ANY`, which means that incoming UDP packets are accepted on any network interface. In some AUTOSAR deployments with multiple network interfaces, it may be necessary to restrict DDS receive sockets to a specific interface so that packets are only received on the intended network.

To enable this behavior, set the `enable_interface_bind` field to `RTI_TRUE` in the UDP transport property (`UDP_InterfaceFactoryProperty`) when registering the UDP transport. When this condition is met, receive sockets are bound to the specific local IP address of the selected interface rather than `INADDR_ANY`.

On AUTOSAR, this binding works as follows: when a receive socket is bound to a specific IP address (not `INADDR_ANY`), the AUTOSAR socket wrapper (`NETIO_AutosarSocket_bind()`) calls `NETIO_AutosarSocket_find_local_addr_id()` to iterate through the configured TcpIp local address IDs (from 0 to `max_local_addr_id`) using `TcpIp_GetIpAddr()`. When a matching IP address is found, the corresponding `TcpIp_LocalAddrIdType` is passed to `TcpIp_Bind()` instead of `TCPIP_LOCALADDRID_ANY`.

**Requirements:**

- The `max_local_addr_id` field in `OSAPI_PortProperty` must be set to the maximum configured IP address identifier in the TcpIp module, so that the address lookup covers all configured addresses.
- The `enable_interface_bind` field must be set to `RTI_TRUE` in the UDP transport property (`UDP_InterfaceFactoryProperty`) when registering the UDP transport.
- `use_socket_owner` must be `TRUE` in `OSAPI_PortProperty`, since the `TcpIp_Bind()` call with a specific local address ID is only performed when the socket owner mechanism is used. This feature is not supported when using SoAd PDU routing (`use_socket_owner = FALSE`).

**Example:**

```
struct UDP_InterfaceFactoryProperty udp_property =
    UDP_InterfaceFactoryProperty_INITIALIZER;

/* Enable interface bind */
udp_property.enable_interface_bind = RTI_TRUE;

/* Configure the allowed interface(s) */
DDS_StringSeq_set_maximum(&udp_property.allow_interface, 1);
DDS_StringSeq_set_length(&udp_property.allow_interface, 1);
*DDS_StringSeq_get_reference(&udp_property.allow_interface, 0) =
    DDS_String_dup("eth0");

/* Register the UDP transport with the modified property */
if (!RT_Registry_register(registry, NETIO_DEFAULT_UDP_NAME,
        UDP_InterfaceFactory_get_interface(),
        (struct RT_ComponentFactoryProperty*)&udp_property, NULL))
{
    /* ERROR */
}
```

### 6.7.3 NoCom/FullCom socket recovery

In some AUTOSAR deployments the communication stack can transition to a *NoCom* (no-communication) state — for example, when a network management event, a diagnostic session, or a bus-off condition causes the TcpIp stack to close all active UDP sockets and release the IP address. When communication is later restored (*FullCom*), *Connext Micro* must re-open and re-bind its receive sockets so that DDS traffic can resume without restarting the ECU.

---

**Note:** This feature applies **only** when the Socket Owner mechanism is used (`use_socket_owner = TRUE` in `OSAPI_PortProperty`). When SoAd PDU routing is used instead, socket lifecycle management is handled directly by SoAd and this recovery path is not needed.

---

This feature is not available in CERT builds (`RTI_CERT`).

*Connext Micro* tracks every active receive socket in an internal table (one entry per bound local port). The recovery sequence has two stages:

1. **Socket closed (NoCom)** — When the TcpIp stack closes a UDP socket it calls the application-registered `TcpIpEvent` callback with the event `TCPIP_UDP_CLOSED`. The application must forward this event to *Connext Micro* by calling `NETIO_Autosar_on_socket_event()`. *Connext Micro* looks up the socket ID in the internal receive-port table and marks the corresponding entry as *closed* (`is_open = FALSE`), so it is eligible for recovery when communication is restored. Send sockets do not need explicit tracking because they are re-bound unconditionally during recovery.
2. **IP address re-assigned (FullCom)** — When the TcpIp stack re-assigns an IP address it calls the application-registered `LocalIpAddrAssignmentChg` callback with `TCPIP_IPADDR_STATE_ASSIGNED`. The application must call `NETIO_Autosar_on_ip_assigned()` for every such event (except the very first one, which is the normal DDS startup path). *Connext Micro* then performs the following steps for each registered UDP interface:
   a. Re-binds all send sockets to the newly assigned local address ID.
   b. Iterates the receive-port table. For each entry that is marked *closed* **and** whose bound IP address matches the newly assigned local address ID:
      i. Deregisters the old socket ID from the TcpIp recvnotify callback table.
      ii. Acquires a fresh socket ID from TcpIp via `TcpIp_GetSocket()`.
      iii. Binds the new socket to the same local port that was active before the NoCom transition.
      iv. Re-registers the *Connext Micro* receive callback (`UDP_Interface_receive_packet`) with the new socket ID.

v. Updates the port entry with the new socket ID and marks it *open* again.

Both functions are serialised with the internal NETIO mutex and are idempotent — calling them when no recovery is needed returns `RTI_TRUE` immediately without modifying any state.

### API

**`NETIO_Autosar_on_socket_event`**

```
Std_ReturnType NETIO_Autosar_on_socket_event(
    TcpIp_SocketIdType socket_id,
    TcpIp_EventType    socket_event);
```

Call this from the `TcpIpEvent` socket-owner callback whenever the TcpIp stack reports a socket event. Only `TCPIP_UDP_CLOSED` events are acted upon; all other event types are ignored.

**`NETIO_Autosar_on_ip_assigned`**

```
Std_ReturnType NETIO_Autosar_on_ip_assigned(
    TcpIp_LocalAddrIdType local_addr_id);
```

Call this from the `LocalIpAddrAssignmentChg` socket-owner callback whenever an IP address transitions to `TCPIP_IPADDR_STATE_ASSIGNED` *after the initial DDS startup*. Pass the `LocalAddrId` reported by the TcpIp stack so that *Connext Micro* can match it against the bound addresses of the affected sockets.

## 6.7.4 How the PIL was built for MicroSAR platforms

This section describes how RTI built the Platform Independent Library (PIL) for MicroSAR platforms.

---

**Note:** *Connext Micro* requires the C-type "double" to be 64 bits. Any compiler option that treats a "double" as a "float" must not be enabled.

---

The following table shows the compiler flags RTI used to create the PIL for MicroSAR platforms:

Table 6.20: PIL Compiler Flags for MicroSAR Platforms

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| tc39xtElfTasking6.3r1 | Static Release | -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O2ROP -DNDEBUG |
| | Static Debug | -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O0 -g2 |
| tc39xtElfTasking6.3r1CERT | Static Release | -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O2ROP -DNDEBUG |
| | Static Debug | -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O0 -g2 |

Table 6.21: Compiler Flag Details

| Flag | Purpose |
|---|---|
| `-Ctc39xb` | Select the TC39xB CPU target |
| `-D__CPU__=tc39xb` | Define CPU type macro (required by MicroSAR headers) |
| `-D__CPU_TC39XB__` | Define CPU-specific macro |
| `--core=tc1.6.2` | Select TriCore architecture core version 1.6.2 |
| `--align=0` | Default alignment (no forced alignment) |
| `-Wc--no-warnings=504` | Suppress warning 504: macro redefined (for ___CPU___) |
| `-Wcp--no-warnings=550` | Suppress warning 550: variable set but never used |
| `--fp-model=c,F,L,N,R,S,T,Z` | Precise floating-point model ensuring 64-bit doubles |
| `--language=-gcc,-volatile,+strings, -kanji` | Language extensions configuration |
| `--iso=99` | ISO C99 standard |
| `--switch=auto` | Automatic switch statement code generation |
| `--compact-max-size=200` | Maximum size for code compaction (reverse inlining) |
| `--error-limit=1` | Stop after first error |
| `--source` | Merge C source as comments in assembly output |
| `--integer-enumeration` | Treat enums as int per ISO C99 |
| `-t4` | Optimize for code size |
| `-Z0` | Disable ___a0 (near addressing) allocation |
| `-Y0` | Disable ___a1 (near addressing) allocation |
| `--default-near-size=0` | Disable near-code optimizations (simplifies MPU config) |
| `-O2ROP` (Release) | Optimization level 2 with R, O, P optimizations |
| `-DNDEBUG` (Release) | Disable debug assertions |
| `-O0` (Debug) | No optimization |
| `-g2` (Debug) | Full debug information |

### 6.7.5 Building the PSL from source for MicroSAR platforms

This section describes how to build your own Platform Support Library (PSL) for MicroSAR.

---

**Note:** *Connext Micro* requires the C-type "double" to be 64 bits. Any compiler option that treats a "double" as a "float" must not be enabled.

---

*Connext Micro* includes support to compile Platform Support Libraries (PSL) for MicroSAR using CMake. Refer to *Setting up the build environment* before continuing with the following instructions.

1. Make sure CMake is in the path.
2. Define the following environment variables:
   - `OSEK_PATH`: Path to the Vector MicroSAR static code (SIP folder).
   - `RTIME_TASKING_PATH`: Path to the Tasking compiler.
3. Enter the following command:

```
rtime-make.bat --target tc39xtElfTasking6.3r1-MICROSAR4 --name tc39xtElfTasking6.3r1-MICROSAR4 --config Release --
↪delete -G "Ninja" --build
```

   ---

   **Note:** `rtime-make` uses the architecture specified with `--target` to determine a few settings needed by *Connext Micro*. Please refer to *Preparing to build* for details.

   ---

4. The *Connext Micro* PSL is available in:

```
RTIMEHOME/lib/tc39xtElfTasking6.3r1-MICROSAR4
```

You must build applications with compatible flags to the PIL and PSL in order to operate with *Connext Micro*; the PSL must also be binary compatible with the PIL. Applications must not specify the `RTI_PSL` or `RTI_PIL` preprocessor definitions. When you build the PSL with `rtime-make`, the `--target` argument automatically adds all the necessary flags for the specified architecture.

The following table shows the compiler flags and required options that RTI used to build the PSL for MicroSAR platforms.

Table 6.22: PSL Compiler Flags for MicroSAR Platforms

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| tc39xtElfTasking6.3r1-MICROSAR4 | Static Release | -D___autosar___=1 -DRTI_ENDIAN_LITTLE=1 -DRTIME_AUTOSAR_MICROSAR=1 -DRTI_AUTOSAR=1 -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O2ROP -DNDEBUG |
| | Static Debug | -D___autosar___=1 -DRTI_ENDIAN_LITTLE=1 -DRTIME_AUTOSAR_MICROSAR=1 -DRTI_AUTOSAR=1 -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O0 -g2 |
| tc39xtElfTasking6.3r1CERT-MICROSAR4 | Static Release | -D___autosar___=1 -DRTI_ENDIAN_LITTLE=1 -DRTIME_AUTOSAR_MICROSAR=1 -DRTI_AUTOSAR=1 -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O2ROP -DNDEBUG |
| | Static Debug | -D___autosar___=1 -DRTI_ENDIAN_LITTLE=1 -DRTIME_AUTOSAR_MICROSAR=1 -DRTI_AUTOSAR=1 -Ctc39xb -D___CPU___=tc39xb -D___CPU_TC39XB___ –core=tc1.6.2 –align=0 -Wc–no-warnings=504 -Wcp–no-warnings=550 –fp-model=c,F,L,N,R,S,T,Z –language=-gcc,-volatile,+strings,-kanji –iso=99 –switch=auto –compact-max-size=200 –error-limit=1 –source –integer-enumeration -t4 -Z0 -Y0 –default-near-size=0 -O0 -g2 |

### 6.7.6 Known issues and limitations

Table 6.23: Known Issues and Limitations

| Issue | Details |
|---|---|
| `enable_thread_safe_heap` set to FALSE | Disabling this property may raise errors since we do not prevent multiple concurrent accesses to the heap. We recommend keeping this set to `TRUE`. |
| Spinlock synchronization not fully tested | The port has not been tested with `sync_type` set to `OSAPI_AUTOSAR_SYNCKIND_SPINLOCK`. Use `OSAPI_AUTOSAR_SYNCKIND_RESOURCES` (the default) for production deployments. |
| `use_socket_owner = FALSE` not fully tested | The port has not been tested with `use_socket_owner` set to FALSE. When `use_socket_owner` is FALSE, the `send_data` function pointer must be provided instead. |
| No dynamic memory deallocation | The AUTOSAR heap implementation does not support freeing memory (`OSAPI_DONT_HAVE_REALLOC`). Once memory is allocated from the static heap, it cannot be returned. Ensure sufficient heap area is provisioned. |
| Single instantiation only | The DDS CDD does not support multiple instantiations. |
| C++ not supported | C++ compilation is excluded (`RTIME_EXCLUDE_CPP`). The AUTOSAR port is C-only. |
| Shared memory transport not supported | Shared memory transport is excluded (`RTIME_EXCLUDE_SHMEM`) on AUTOSAR platforms. |
| Threads not supported | AUTOSAR does not provide a standard thread API. The macro `OSAPI_NO_THREADS` is set to 1. All DDS processing is done within AUTOSAR runnables/tasks. |
| Flow controller and fragmentation not supported | The DDS flow controller and fragmentation features are not supported on AUTOSAR platforms. |

### 6.7.7 Example

For an in-depth example of how to successfully integrate *Connext Micro* with AUTOSAR and understand the resulting architecture, refer to *Appendix A: AUTOSAR Integration Example* and *Appendix B: AUTOSAR Architecture Example*.

## 6.8 macOS Platforms

The following table shows the currently supported macOS platforms.

Table 6.24: Supported macOS Platforms

| OS | Version | CPU | Network Stack | Toolchain | Architecture PIL | Architecture PSL |
|---|---|---|---|---|---|---|
| macOS | 14 | arm64 | OS Default | clang 15.0 | armv8leMachOclang15.0<br>armv8leMachOclang15.0CERT | armv8leMachOclang15.0-Darwin23<br>armv8leMachOclang15.0CERT-Darwin23 |

### 6.8.1 How the PIL was built for macOS platforms

This section describes how RTI built the Platform Independent Library (PIL) for macOS.

The following table shows the compiler flags RTI used to create the PIL for macOS platforms:

Table 6.25: PIL Compiler Flags for macOS Platforms

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv8leMachOclang15.0 | Static Release | **C Flags:** -std=c99 -nostdinc -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG **C++ Flags:** -fstrict-aliasing -fsigned-char -O2 -fPIC -DNDEBUG |
| | Shared Release | **C Flags:** -std=c99 -nostdinc -O2 -fstrict-aliasing -fsigned-char -fPIC -DNDEBUG **C++ Flags:** -fstrict-aliasing -fsigned-char -O2 -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC **C++ Flags:** -fsigned-char -fstrict-aliasing -g -fPIC |
| | Shared Debug | **C Flags:** -std=c99 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC **C++ Flags:** -fsigned-char -fstrict-aliasing -g -fPIC |

continues on next page

Table 6.25 – continued from previous page

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv8leMachOclang15.0CERT | Static Release | **C Flags:** -std=c99 -O2 -fsigned-char -nostdinc -fstrict-aliasing -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fsigned-char -nostdinc -fstrict-aliasing -g -fPIC |

**Warning:** The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, *Connext Micro* makes one direct call to the C library API `qsort`. In addition, CLANG may insert direct calls to GLIBC functions and other required functions, such as default C++ constructors and destructors. For this reason, it is necessary to use a CLANG version that is compatible with the CLANG version used to build the platform independent libraries, or provide a C library with an implementation of the required functions. Future versions of *RTI Connext Micro* will remove these dependencies.

### 6.8.2 Building the PSL from source for macOS platforms

Refer to *Building the PSL* for instructions on how to build your own Platform Support Library (PSL) for macOS platforms.

The following table shows the compiler flags and required options that RTI used to build the PSL for macOS platforms. When you build the PSL with `rtime-make`, the `--target` argument automatically adds all the necessary flags for the specified architecture.

Table 6.26: PSL Compiler Flags for macOS Platforms

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv8leMachOclang15.0-Darwin23 | Static Release | **C Flags:** -std=c99 -fsigned-char -O2 -fstrict-aliasing -O2 -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC -DNDEBUG **C++ Flags:** -fstrict-aliasing -fsigned-char -O2 -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC -DNDEBUG |

continues on next page

Table 6.26 – continued from previous page

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Shared Release | **C Flags:** -std=c99 -fsigned-char -O2 -fstrict-aliasing -O2 -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC -DNDEBUG **C++ Flags:** -fstrict-aliasing -fsigned-char -O2 -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fsigned-char -fstrict-aliasing -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC **C++ Flags:** -fsigned-char -fstrict-aliasing -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC |
| | Shared Debug | **C Flags:** -std=c99 -fsigned-char -fstrict-aliasing -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC **C++ Flags:** -fsigned-char -fstrict-aliasing -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC |
| armv8leMachOclang15.0CERT-Darwin23 | Static Release | **C Flags:** -std=c99 -fsigned-char -O2 -fstrict-aliasing -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -std=c99 -fsigned-char -fstrict-aliasing -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX14.5.sdk -mmacosx-version-min=11 -fPIC |

## 6.9 QNX Platforms

The following table shows the currently supported QNX platforms.

Table 6.27: Supported QNX Platforms

| OS | Version | CPU | Network Stack | Toolchain | Architecture PIL | Architecture PSL |
|---|---|---|---|---|---|---|
| QNX | 7.1 | ARMv8 (64-bit) | OS Default: io-pkt | qcc_gpp8.3.0 | armv8leElfqnx_qcc8.3.0<br>armv8leElfqnx_qcc8.3.0CERT | armv8leElfqnx_qcc8.3.0-QNX7.1<br>armv8leElfqnx_qcc8.3.0CERT-QNX7.1 |
| QNX | 8.0 | ARMv8 (64-bit) | OS Default: io-sock | qcc_cxx12.2.0 | armv8leElfqnx_qcc12.2.0<br>armv8leElfqnx_qcc12.2.0CERT | armv8leElfqnx_qcc12.2.0-QNX8.0<br>armv8leElfqnx_qcc12.2.0CERT-QNX8.0 |
| QOS (QNX OS for Safety) | 2.2.1 | ARMv8 (64-bit) | OS Default: io-pkt | qcc_gpp8.3.0 | armv8leElfqcc8.3.0<br>armv8leElfqcc8.3.0CERT | armv8leElfqcc8.3.0-QOS2.2.1<br>armv8leElfqcc8.3.0CERT-QOS2.2.1 |

### 6.9.1 How the PIL was built for QNX platforms

This section describes how RTI built the Platform Independent Library (PIL) for QNX platforms.

The following table shows the compiler flags RTI used to create the PIL for QNX platforms:

Table 6.28: PIL Compiler Flags for QNX Platforms

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv8leElfqnx_qcc8.3.0 | Static Release | **C Flags:**-nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -DNDEBUG **C++ Flags:**-nostdinc -fsigned-char -O2 -Vgcc/8.3.0,gcc_ntoaarch64le -fPIC -lang-c++ -DNDEBUG |
| | Shared Release | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -DNDEBUG **C++ Flags:** -nostdinc -fsigned-char -O2 -Vgcc/8.3.0,gcc_ntoaarch64le -fPIC -lang-c++ -DNDEBUG |

continues on next page

Table 6.28 – continued from previous page

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Static Debug | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC **C++ Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC -lang-c++ |
| | Shared Debug | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC **C++ Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC -lang-c++ |
| armv8leElfqnx_qcc8.3.0CERT | Static Release | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -DNDEBUG |
| | Static Debug | **C Flags:**-nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC |
| armv8leElfqnx_qcc12.2.0 | Static Release | **C Flags:** -nostdinc -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC **C++ Flags:** -lang-c++ -nostdinc -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC |
| | Shared Release | **C Flags:** -nostdinc -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC **C++ Flags:** -lang-c++ -nostdinc -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC |
| | Static Debug | **C Flags:** -nostdinc -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC **C++ Flags:** -lang-c++ -nostdinc -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC |
| | Shared Debug | **C Flags:** -nostdinc -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC **C++ Flags:** -lang-c++ -nostdinc -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC |
| armv8leElfqnx_qcc12.2.0CERT | Static Release | **C Flags:** -nostdinc -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC |
| | Static Debug | **C Flags:** -nostdinc -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC |
| armv8leElfqcc8.3.0 | Static Release | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -DNDEBUG **C++ Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -lang-c++ -DNDEBUG |

continues on next page

Table 6.28 – continued from previous page

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Shared Release | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -DNDEBUG **C++ Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -lang-c++ -DNDEBUG |
| | Static Debug | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC **C++ Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC -lang-c++ |
| | Shared Debug | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC **C++ Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC -lang-c++ |
| armv8leElfqcc8.3.0CERT | Static Release | **C Flags:** -nostdinc -fsigned-char -O2 -Vgcc/8.3.0,gcc_ntoaarch64le -O2 -fPIC -DNDEBUG |
| | Static Debug | **C Flags:** -nostdinc -fsigned-char -Vgcc/8.3.0,gcc_ntoaarch64le -g -fPIC |

**Warning:** The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, *Connext Micro* makes one direct call to the C library API `qsort`. In addition, QCC may insert direct calls to GLIBC functions and other required functions, such as default C++ constructors and destructors. For this reason, it is necessary to use a QCC version that is compatible with the QCC version used to build the platform independent libraries, or provide a C library with an implementation of the required functions. Future versions of *RTI Connext Micro* will remove these dependencies.

### 6.9.2 Building the PSL from source for QNX platforms

Refer to *Building the PSL* for instructions on how to build your own Platform Support Library (PSL) for QNX platforms.

The following table shows the compiler flags and required options that RTI used to build the PSL for QNX platforms. When you build the PSL with `rtime-make`, the `--target` argument automatically adds all the necessary flags for the specified architecture.

Table 6.29: PSL Compiler Flags for QNX Platforms

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| armv8leElfqnx_qcc8.3.0-QNX7.1 | Static Release | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -DNDEBUG<br>**C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -O2 -lang-c++ -DNDEBUG |
| | Static Debug | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -DNDEBUG<br>**C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -O2 -lang-c++ -DNDEBUG |
| | Shared Release | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -DNDEBUG<br>**C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -O2 -lang-c++ -DNDEBUG |
| | Shared Debug | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -DNDEBUG<br>**C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -O2 -lang-c++ -DNDEBUG |
| armv8leElfqnx_qcc8.3.0CERT-QNX7.1 | Static Release | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -O2 -DNDEBUG |
| | Static Debug | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -g |
| armv8leElfqnx_qcc12.2.0-QNX8.0 | Static Release | **C Flags:** -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC<br>**C++ Flags:** -lang-c++ -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC |
| | Static Debug | **C Flags:** -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC<br>**C++ Flags:** -lang-c++ -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC |
| | Shared Release | **C Flags:** -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC<br>**C++ Flags:** -lang-c++ -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC |

continues on next page

Table 6.29 – continued from previous page

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Shared Debug | **C Flags:** -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC **C++ Flags:** -lang-c++ -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC |
| armv8leElfqnx_qcc12.2.0CERT-QNX8.0 | Static Release | **C Flags:** -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -O2 -fPIC |
| | Static Debug | **C Flags:** -DNDEBUG -Vgcc/12.2.0,gcc_ntoaarch64le -fsigned-char -g -fPIC |
| armv8leElfqcc8.3.0-QOS2.2.1 | Static Release | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -O2 -DNDEBUG **C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -O2 -lang-c++ -DNDEBUG |
| | Static Debug | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -g **C++ Flags:**-Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -g -lang-c++ |
| | Shared Release | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -O2 -DNDEBUG **C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -O2 -lang-c++ -DNDEBUG |
| | Shared Debug | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -g **C++ Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fsigned-char -g -lang-c++ |
| armv8leElfqcc8.3.0CERT-QOS2.2.1 | Static Release | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -DNDEBUG |
| | Static Debug | **C Flags:** -Vgcc/8.3.0,gcc_ntoaarch64le -fsigned-char -Y_gpp -g |

## 6.10 Windows Platforms

The following table shows the currently supported Windows platforms.

Table 6.30: Supported Windows Platforms

| OS | Version | CPU | Network Stack | Toolchain | Architecture PIL | Architecture PSL |
|---|---|---|---|---|---|---|
| Windows | 10 | x64 | OS Default | Visual Studio 2017 | x86_64lePEvs2017<br>x86_64lePEvs2017CERT | x86_64lePEvs2017-Win10<br>x86_64leP-Evs2017CERT-Win10 |

### 6.10.1 How the PIL was built for Windows platforms

This section describes how RTI built the Platform Independent Library (PIL) for Windows platforms.

The following table shows the compiler flags RTI used to create the PIL for Windows platforms:

Table 6.31: PIL Compiler Flags for Windows Platforms

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| x86_64lePEvs2017 | Static/Shared Release | **C Flags:** _WINDOWS;WIN32;_WIN-DOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SE-CURE_NO_WARNINGS; **C++ Flags:** _WINDOWS;WIN32;_WIN-DOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SE-CURE_NO_WARNINGS; |
| | Static/Shared Debug | **C Flags:** _DEBUG;_WINDOWS;WIN32;_WIN-DOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SE-CURE_NO_WARNINGS; **C++ Flags:** _DEBUG;_WIN-DOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600 CURE_NO_WARNINGS; |

continues on next page

Table 6.31 – continued from previous page

| Architecture PIL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| x86_64lePEvs2017CERT | Static Release | **C Flags:** _WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; |
| | Static Debug | **C Flags:** _DEBUG;_WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; |

**Warning:** The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, *Connext Micro* makes one direct call to the C library API `qsort`. In addition, MSVS may insert direct calls to C library functions and other required functions, such as default C++ constructors and destructors. For this reason it is necessary to use an MSVS version that is compatible with the MSVS version used to build the platform independent libraries, or provide a C library with an implementation of the required functions. Future versions of *RTI Connext Micro* will remove these dependencies.

### 6.10.2 Building the PSL from source for Windows platforms

Refer to *Building the PSL* for instructions on how to build your own Platform Support Library (PSL) for Windows platforms.

The following table shows the compiler flags and required options that RTI used to build the PSL for Windows platforms. When you build the PSL with `rtime-make`, the `--target` argument automatically adds all the necessary flags for the specified architecture.

Table 6.32: PSL Compiler Flags for Windows Platforms

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| x86_64lePEvs2017-Win10 | Static/Shared Release | **C Flags:** _WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; **C++ Flags:** _WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; |

continues on next page

Table 6.32 – continued from previous page

| Architecture PSL | Library Format | Compiler Flags Used by RTI |
|---|---|---|
| | Static/Shared Debug | **C Flags:** _DEBUG;_WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; **C++ Flags:** _DEBUG;_WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; |
| x86_64lePEvs2017CERT-Win10 | Static Release | **C Flags:** _WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; |
| | Static Debug | **C Flags:** _DEBUG;_WINDOWS;WIN32;_WINDOWS;WIN32_LEAN_AND_MEAN;_WIN32_WINNT=0x0600;_CRT_SECURE_NO_WARNINGS; |

# Chapter 7

# Connext Tools

In general, *Connext Micro* is compatible with RTI tools and other products. The following sections provide additional information for each product.

## 7.1 Admin Console

Admin Console can discover and display *Connext Micro* applications that use full dynamic discovery (DPDE). When using static discovery (DPSE), it is required to use the Limited Bandwidth Endpoint Discovery (LBED) that is available as a separate product for *Connext Professional*. With the library a configuration file with the discovery configuration must be provided (just as in the case for products such as Routing Service, etc.). This is provided through the QoS XML file.

Data can be visualized from *Connext Micro DataWriters*. Keep in mind that *Connext Micro* does not currently distribute type information and the type information has to be provided through an XML file using the "Create Subscription" dialog. Unlike some other products, this information cannot be provided through the QoS XML file. To provide the data types to Admin Console, first run the code generator with the `-convertToXml` option:

```
rtiddsgen -convertToXml <file>
```

Then click on the "Load Data Types from XML file" hyperlink in the "Create Subscription" dialog and add the generated IDL file.

Other Features Supported:

- Match analysis is supported.
- Discovery-based QoS are shown.

The following resource-limits in *Connext Micro* must be incremented as follows when using Admin Console:

- Add 24 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_reader_allocation
- Add 24 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_writer_allocation
- Add 1 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_allocation

- Add 1 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_allocation if data-visualization is used

*Connext Micro* does not currently support any administration capabilities or services, and does not match with the Admin Console *DataReaders* and *DataWriters*. However, if matching *DataReaders* and *DataWriters* are created, e.g., by the application, the following resource must be updated:

- Add 48 to DDS_DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_allocation

## 7.2 Distributed Logger

This product is not supported by *Connext Micro*.

## 7.3 LabVIEW

The LabVIEW toolkit uses *Connext Professional*, and it must be configured as any other *Connext Professional* application. A possible option is to use the builtin *Connext Professional* profile: BuiltinQosLib::Generic.ConnextMicroCompatibility.

## 7.4 Monitor

This product is not supported by *Connext Micro*.

## 7.5 Recording Service

### 7.5.1 RTI Recorder

RTI Recorder is compatible with *Connext Micro* in the following ways:

- If static endpoint discovery is used, Recorder is compatible starting with version 5.1.0.3 and onwards.
- If dynamic endpoint discovery is used, Recorder is compatible with *Connext Micro* the same way it is with any other DDS application.
- In both cases, type information has to be provided via XML. Read Recording Data with RTI Connext Micro for more information.

### 7.5.2 RTI Replay

RTI Replay is compatible with *Connext Micro* in the following ways:

- If static endpoint discovery is used, Replay is compatible starting with version 5.1.0.3 and onwards.
- If dynamic endpoint discovery is used, Replay is compatible with *Connext Micro* the same way it is with any other DDS application.
- In both cases, type information has to be provided via XML. Read Recording Data with RTI Connext Micro for more information on how to convert from IDL to XML.

### 7.5.3 RTI Converter

Databases recorded with *Connext Micro* contains no type information in the DCPSPublication table, but the type information can be provided via XML. Read Recording Data with RTI Connext Micro for more information on how to convert from IDL to XML.

## 7.6 Wireshark

Wireshark fully supports *Connext Micro*.

## 7.7 Persistence Service

*Connext Micro* only supports VOLATILE and TRANSIENT_LOCAL durability and does not support the use of Persistence Service.

## 7.8 Application Generation Using XML

An application defined in XML can be shared between *Connext Micro* and *Connext Professional*, with the limitations documented in *Application Generation Using XML*.

# Chapter 8

# Migrating from Connext Professional

In some cases, it may be necessary to write an application that is compiled against both *Connext Micro*, *Connext Cert*, and *Connext Professional*. In general this is not easy to do because *Connext Micro* and *Connext Cert* support a very limited set of features compared to *Connext Professional*. However, while *Connext Cert* is subset of *Connext Micro*, it is relatively easy to write applications that support both.

Due to the nature of the DDS API and the philosophy of declaring behavior through QoS profiles instead of using different APIs, it may be possible to share common code. In particular, *Connext Professional* supports configuration through QoS profile files, which eases the job of writing portable code.

Note that *Connext Professional* supports many extended APIs that are not covered by the DDS specification; for example, APIs that create DDS *Entities* based on QoS profiles.

## 8.1 Supported DDS Features

*Connext Micro* supports a subset of the DDS DCPS standard. For a detailed list of supported *Entities* and QoS policies, please refer to the C API Reference and C++ API Reference.

## 8.2 Development Environment

There are no conflicts between *Connext Micro* and *Connext Professional* with respect to library names, header files, etc. It is advisable to keep the two installations separate, which is the normal case.

*Connext Micro* uses the environment variable RTIMEHOME to locate the root of the *Connext Micro* installation.

*Connext Professional* uses the environment variable NDDSHOME to locate the root of the *Connext Professional* installation.

## 8.3 Non-standard APIs

The DDS specification omits many APIs and policies necessary to configure a DDS application, such as transport, discovery, memory, logging, etc. In general, *Connext Micro* and *Connext Professional* do not share APIs for these functions.

It is recommended to configure *Connext Professional* using QoS profiles as much as possible.

## 8.4 QoS Policies

QoS policies defined by the DDS standard behave the same between *Connext Micro* and *Connext Professional*. However, note that *Connext Micro* does not always support all the values for a policy and in particular unlimited resources are not supported.

Unsupported QoS policies are the most likely reason for not being able to switch between *Connext Micro* and *Connext Professional*.

## 8.5 Standard APIs

APIs that are defined by the standard behave the same between *Connext Micro* and *Connext Professional*.

## 8.6 IDL Files

*Connext Micro* and *Connext Professional* use the same IDL compiler (rtiddsgen) and *Connext Micro* typically ships with the latest version. However, *Connext Micro* and *Connext Professional* use different templates to generate code and it is not possible to share the generated code. Thus, while the same IDL can be used, the generated output must be saved in different locations.

# Chapter 9

# API Reference

*RTI Connext Micro* features API support for C and C++. Select the appropriate language below in order to access the corresponding API Reference HTML documentation.

- C API Reference
- C++ API Reference

# Chapter 10

# Release Notes

## 10.1 Compatibility

*Connext Micro* 4.3.0 requires the following minimum versions of *Connext* products for compatibility:

- *Connext Professional* 7.3.1 and above[1]
- *Connext Drive* 4.0.1 and above[1]

Refer to *Migrating from Connext Professional* and the *API Reference* for more details about interoperable APIs, QoS policies, and *Connext* tools.

## 10.2 Supported Platforms and Programming Languages

*Connext Micro* supports the C and traditional C++ language bindings.

Note that RTI only tests on a subset of the possible combinations of OSs and CPUs. Please refer to the following table for a list of specific platforms and the specific configurations that are tested by RTI.

For more information on the library types (PIL, PSL, and integrated) that RTI provides, refer to *Library descriptions*.

PIL

RTI provides PILs for the platforms listed below. Architecture abbreviations utilize the following format:

```
{cpu}{compiler}{profile}
```

- `{cpu}`: the CPU that the library was compiled for.
- `{compiler}`: the compiler used to build the library.
- `{profile}`: CERT if the library was built to be Cert-compatible; otherwise empty.

[1] *Connext Micro* 4.3.0 is not guaranteed to be compatible with all future releases of *Connext* products. Consult the **Downloads** page on support.rti.com for up-to-date compatibility information.

Table 10.1: Supported Platforms (PIL)

| OS | CPU | Compiler | RTI Architecture Abbreviations |
|---|---|---|---|
| Windows® 10 | x64 | VS 2017 | x86_64lePEvs2017<br>x86_64lePEvs2017CERT |
| macOS® 14 | arm64 | clang 15.0 | armv8leMachOclang15.0<br>armv8leMachOclang15.0CERT |
| Ubuntu® 24.04 LTS | x64 | gcc 13.3.0 | x86_64leElfgcc13.3.0<br>x86_64leElfgcc13.3.0CERT |
| Ubuntu® 24.04 LTS | ARMv8 (64-bit) | gcc 13.3.0 | armv8leElfgcc13.3.0<br>armv8leElfgcc13.3.0CERT |
| Yocto 5.15.96 | ARMv8-A (64-bit) | gcc 11.3.1 | armv8aleElfgcc11.3.1<br>armv8aleElfgcc11.3.1CERT |
| QNX® 7.1 | ARMv8 (64-bit) | qcc_gpp8.3.0 | armv8leElfqnx_qcc8.3.0<br>armv8leElfqnx_qcc8.3.0CERT |
| QNX® 8.0 | ARMv8 (64-bit) | qcc_cxx12.2.0 | armv8leElfqnx_qcc12.2.0<br>armv8leElfqnx_qcc12.2.0CERT |
| QOS 2.2.1 (QNX OS for Safety) | ARMv8 (64-bit) | qcc_gpp8.3.0 | armv8leElfqcc8.3.0<br>armv8leElfqcc8.3.0CERT |
| FreeRTOS® 10.0.0 | Armv7E-M | gcc 10.3.1 | armv7emleElfgcc10.3.1<br>armv7emleElfgcc10.3.1CERT |
| MicroSAR 4.2.2 | TC397 | Tasking 6.3r1p7 | tc39xtElfTasking6.3r1<br>tc39xtElfTasking6.3r1CERT |

PSL

RTI provides PSLs for the platforms listed below. Architecture abbreviations utilize the following format:

```
{cpu}{compiler}{profile}-{OS}
```

- `{cpu}`: the CPU that the library was compiled for.
- `{compiler}`: the compiler used to build the library.
- `{profile}`: CERT if the library was built to be Cert-compatible; otherwise empty.
- `{OS}`: The operating system that the PSL was compiled for.

Table 10.2: Supported Platforms (PSL)

| OS | CPU | Compiler | RTI Architecture Abbreviations |
|---|---|---|---|
| Windows® 10 | x64 | VS 2017 | x86_64lePEvs2017-Win10<br>x86_64lePEvs2017CERT-Win10 |
| macOS® 14 | arm64 | clang 15.0 | armv8leMachOclang15.0-Darwin23<br>armv8leMachOclang15.0CERT-Darwin23 |
| Ubuntu® 24.04 LTS | x64 | gcc 13.3.0 | x86_64leElfgcc13.3.0-Linux6<br>x86_64leElfgcc13.3.0CERT-Linux6 |

continues on next page

Table 10.2 – continued from previous page

| OS | CPU | Compiler | RTI Architecture Abbreviations |
|---|---|---|---|
| Ubuntu® 24.04 LTS | ARMv8 (64-bit) | gcc 13.3.0 | armv8leElfgcc13.3.0-Linux6<br>armv8leElfgcc13.3.0CERT-Linux6 |
| Yocto 5.15.96 | ARMv8-A (64-bit) | gcc 11.3.1 | armv8aleElfgcc11.3.1-Linux5<br>armv8aleElfgcc11.3.1CERT-Linux5 |
| QNX® 7.1 | ARMv8 (64-bit) | qcc_gpp8.3.0 | armv8leElfqnx_qcc8.3.0-QNX7.1<br>armv8leElfqnx_qcc8.3.0CERT-QNX7.1 |
| QNX® 8.0 | ARMv8 (64-bit) | qcc_cxx12.2.0 | armv8leElfqnx_qcc12.2.0-QNX8.0<br>armv8leElfqnx_qcc12.2.0CERT-QNX8.0 |
| QOS 2.2.1 (QNX OS for Safety) | ARMv8 (64-bit) | qcc_gpp8.3.0 | armv8leElfqcc8.3.0-QOS2.2.1<br>armv8leElfqcc8.3.0CERT-QOS2.2.1 |
| FreeRTOS® 10.0.0 | Armv7E-M | gcc 10.3.1 | armv7em-leElfgcc10.3.1-FreeRTOS10.0<br>armv7em-leElfgcc10.3.1CERT-FreeRTOS10.0 |
| MicroSAR 4.2.2 | TC397 | Tasking 6.3r1p7 | tc39xtElfTasking6.3r1-MICROSAR4<br>tc39xtElfTask-ing6.3r1CERT-MICROSAR4 |

Integrated

RTI does not provide integrated libraries for this release.

## 10.3 What's New in 4.3.0

*RTI Connext Micro* 4.3.0 is an LTS release. See the Connext Releases page on the RTI website for more information on RTI's software release model.

The following features are new since *Connext Micro* 4.2.0.

### 10.3.1 Build Connext Micro applications for MicroSAR platforms

This release adds support for MicroSAR platforms. Refer to *MicroSAR Platforms* for more details.

### 10.3.2 Generate AUTOSAR code with Micro Application Generator

This release adds several new features to Micro Application Generator (MAG) which, taken together, enable you to generate AUTOSAR applications from XML configuration files. Refer to *Generating AUTOSAR Applications* for a walkthrough of the process. These new features include:

#### DDS System XML model support

MAG now supports the DDS System XML schema flow for application generation, which extends the capabilities of the standard XML configuration model. Refer to *Define your system in DDS system XML* for details.

#### Deployment-targeted code generation

A new `-deployment <name>` command-line option lets you generate code for a single deployment instance selected from a deployment scenario defined in the XML. Refer to *MAG Command-Line Options* for more information.

#### Application type selection

A new `-applicationType <type>` option controls the structure and templates used in code generation. Valid values are `micro4`, `cert`, and `autosar_cdd`. Refer to *MAG Command-Line Options* for more information.

> **Attention:** Currently, only `autosar_cdd` is fully supported for deployment-based code generation; see the *Known Issues* section. This type generates AUTOSAR-compliant code structure, including adapter layer separation and AUTOSAR resource configuration templates. Other application types may be supported in future releases.

### 10.3.3 Speed up initialization on AUTOSAR systems with configurable thread safety

This release adds a new AUTOSAR port property, `enable_thread_safe_heap`, that allows the application to control whether heap allocation is protected by a mutex. When enabled, OSAPI_Heap_allocate_buffer() acquires and releases a mutex around memory allocation to ensure thread safety. When disabled, the mutex is neither created nor used, reducing overhead in single-threaded or non-preemptive environments.

### 10.3.4 Generate native applications with deployment-aware code generation

> **Attention:** This is an experimental feature in *Connext Micro* 4.3.0. It is not guaranteed to be consistent or supported and should not be used in production.
>
> Refer to *Experimental Features* for more information.

This release adds a new `-deployment <name>` command-line option to Micro Application Generator that lets you generate code for a specific deployment instance, selected from a deployment scenario defined in the XML. You can generate code for native *Connext Micro* applications with the `micro4` option, as described in *Generating native DDS applications with deployment-aware code generation*.

### 10.3.5 Content filtering no longer experimental

*Connext Micro* 4.2.0 introduced the content filtering feature, enabling *DataReaders* to filter incoming sample data. Instructions for implementing this feature are provided in the *Content Filtering* section.

In release 4.2.0, this feature was experimental, but in release 4.3.0, content filtering is ready for production use.

### 10.3.6 Stabilized builtin DDS SQL Content Filter configuration

This release moves the configuration options for the builtin DDS SQL content filter from the `DDS_FilterResourceLimits` to a new `DDS_SqlFilterProperty`. See *Content Filtering* for how to configure the filter with this new property.

### 10.3.7 Expanded SQL content filter capabilities and improved robustness

This release extends the syntax accepted by *Connext Micro*'s built-in SQL content filter and improves its robustness, including support in SQL filter expressions for nested struct field references with dotted notation and typedef alias resolution. The SQL parser and evaluator were also improved to better handle edge conditions relating to unsigned integers, malformed literals, and incompatible type comparisons. See *Content Filtering* for more details.

### 10.3.8 Reduce network and CPU usage with custom content filters

This release builds on the addition of DDS SQL filter support in *Connext Micro* 4.2.0 by allowing the use of custom content filters. Content filtering can be used to reduce network and CPU usage by only sending samples over the network which reader's are interested in. See *Content Filtering* for details on how to use this feature.

### 10.3.9 Restart discovery after CPU suspension

This release adds the ability for a *DomainParticipant* to restart the discovery process during runtime. This feature allows your application to return to a known state after suspending and resuming a CPU, among other use cases. See *Restarting Discovery* for more information.

### 10.3.10 Prioritize data with the TRANSPORT_PRIORITY QoS policy

This release introduces support for the TRANSPORT_PRIORITY QoS policy in *Connext Micro*. This feature allows applications to assign priorities to outgoing network traffic so that the underlying operating system (OS) and network infrastructure can treat higher-priority messages preferentially.

When enabled, the transport priority value specified in the QoS policy is mapped to the network packet's priority field, such as the IP Type of Service (TOS) or Differentiated Services Code Point

(DSCP) field. This allows applications to influence packet scheduling and prioritization in the network stack, which can be useful for latency-sensitive data flows.

*Connext Micro* currently supports this feature on UDP transports. You can configure the mapping from DDS transport priority values to network packet priority through transport properties.

---

**Note:** The actual prioritization behavior depends on the operating system and network configuration. *Connext Micro* sets the packet priority value, but enforcement of traffic prioritization is handled by the OS networking stack and network infrastructure.

---

For more information on this feature, refer to the *Transport Priority* section.

### 10.3.11 Integrate rotating or on-demand credentials with Lightweight Security Plugin

This release introduces the *passphrase tracker*, a mechanism that enables runtime passphrase rotation for the *Lightweight Security Plugin*. Previously, the pre-shared key (PSK) passphrase was fixed at initialization time. With the *passphrase tracker*, applications can now update the active passphrase without restarting, allowing integration with key management workflows that rotate credentials on a schedule or on demand.

The *passphrase tracker* is designed around a pluggable provider model. Applications supply a provider implementation at configuration time, and the tracker automatically notifies the *Lightweight Security Plugin* whenever a new passphrase is available, re-deriving session keys transparently.

*Connext Micro* ships a reference implementation: the *file passtracker*, available in the `rti_me_psl` PSL library. This implementation monitors a passphrase file on the file system and propagates any changes to the *Lightweight Security Plugin* automatically. The polling interval and startup behavior are configurable.

---

**Note:** The *file passtracker* is only available on POSIX and Windows systems. Applications targeting other platforms or requiring a different passphrase source can provide a custom PSL implementation through the *passphrase tracker* provider interface.

---

For more information, refer to the *Lightweight Security Plugin* section.

### 10.3.12 Reduced heap memory usage

This version of *Connext Micro* uses less dynamically allocated memory on average than the previous release. Refer to the RTI Connext Performance Benchmarks on RTI Community for more details.

### 10.3.13 Resume DDS traffic without restarting ECUs with socket recovery

This release adds automatic socket recovery support for AUTOSAR NoCom/FullCom transitions. When the TcpIp stack closes sockets during a NoCom event (e.g., network management, diagnostic session, or bus-off), *Connext Micro* now tracks affected receive sockets and automatically re-binds them when communication is restored (FullCom). This allows DDS traffic to resume without requiring an ECU restart.

See *NoCom/FullCom socket recovery* for more information. This feature is not available in CERT builds.

### 10.3.14 Default UDP transport name changed

Previously, Micro Application Generator (MAG) automatically appended an index to the UDP transport name; this has been disabled by default. It can be re-enabled by using the new command line parameter `-incrementUdpName`. The default name for the UDP transport name has also been changed from `udp` to `udpv4`. See *Application Generation Using XML* for more information.

### 10.3.15 Streamlined default MAG messages

Previously, running `rtiddsmag` without specifying verbosity (`-verbosity <number>`) would default to info level, which prints informative messages as well as warnings. This has been changed to `warning` level by default. See *MAG Command-Line Options* for more information.

### 10.3.16 Reduced runtime memory usage of MAG-generated code

Starting with this release, all configuration structures generated by Micro Application Generator are now `const`. This may reduce RAM usage significantly if read-only sections are not loaded in memory and instead remain in NVRAM.

### 10.3.17 Third-party and open source software changes

The following third-party software used by *Micro Application Generator* has been upgraded:

Table 10.3: MAG Third-Party Software Upgrades

| Third-Party Software | Old Version | New Version |
|---|---|---|
| Apache Log4j | 2.17.1 | 2.25.3 |
| JAXB Runtime | 2.3.1 | 2.3.9 |

The following third-party software used by *rtiddsgen* has also been upgraded:

Table 10.4: RTIDDSGEN Third-Party Software Upgrades

| Third-Party Software | Old Version | New Version |
|---|---|---|
| Apache Log4j | 2.17.1 | 2.25.3 |

For more information, see the *Third-Party and Open Source Software* section.

## 10.4 What's Fixed in 4.3.0

The following are fixes since *Connext Micro* 4.2.0.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

### 10.4.1 Discovery

#### [Major] DomainParticipants did not use initial announcement configuration correctly

When a new a remote *DomainParticipant* was discovered, the *DomainParticipant* did not use the initial announcement count and period to send new data(p).

[RTI Issue ID MICRO-11874]

#### [Minor] Periodic participant announcements not sent at expected rate

Due to clock drift, periodic participant announcements were not sent at the expected rate on POSIX and Windows systems.

[RTI Issue ID MICRO-11805]

#### [Trivial] Possible error when DPSE and participant_discovery_by_name are enabled

When DPSE and `participant_discovery_by_name` were enabled, the following benign error may have been logged:

```
ERROR: ModuleID=10 Errcode=48 X=1 E=0 T=1
```

[RTI Issue ID MICRO-13318]

### 10.4.2 Usability

#### [Major] Network interfaces could not have non-ASCII characters in their name

If any of the adapter names contained one or more non-ASCII characters in a Windows system, creating associated *DomainParticipants* failed and returned `ModuleID=7 Errcode=813 X = 1 E = 0 T = 1`.

[RTI Issue ID MICRO-11127]

### 10.4.3 Transports

#### [Major] Zero Copy v1 limited the total number of shared memory segments to 500

When using the Zero Copy v1 transport, the total number of shared memory segments per *DomainParticipant* was incorrectly limited to 500.

[RTI Issue ID MICRO-3489]

#### [Trivial] UDP registration documented incorrectly

The *Connext Micro* documentation stated incorrectly that the UDP transport was automatically registered.

[RTI Issue ID MICRO-11445]

#### [Trivial] Internal UDP pings failed to send

Internal UDP pings failed to send, which would cause *Connext Micro* to log the following warning on startup:

```
WARNING: ModuleID=4 Errcode=235
```

[RTI Issue ID MICRO-8929]

### 10.4.4 Reliability Protocol and Wire Representation

#### [Minor] DataWriter resent already-acknowledged samples when responding to ACKNACK on a lossy network

When a GAP message was lost on the wire, a *DataWriter* could resend a sample that the *DataReader* had already acknowledged, producing spurious log warnings on the *DataReader*.

[RTI Issue ID MICRO-11946]

### 10.4.5 Callbacks and Waitsets

#### [Major] on_before callbacks could cause DataReader to stop receiving data

Filtering samples with `on_before` callbacks could filter out samples from batched *DataWriters,* which could eventually cause *DataReaders* to stop receiving data.

[RTI Issue ID MICRO-1981]

#### [Minor] DataWriter did not invoke on_offered_incompatible_qos listener when data representations were incompatible

When a *DataWriter* and a *DataReader* had incompatible `DATA_REPRESENTATION` QoS policies (for example, CDR v1 vs. CDR v2), the *DataWriter* did not invoke its `on_offered_incompatible_qos` listener. The endpoints did not match, but the writer-side notification was silently skipped.

[RTI Issue ID MICRO-12033]

### 10.4.6 Security Plugins SDK

#### [Minor] Incorrect block count calculation for key rotation

The block count calculation used to determine session key rotation in the PSK interceptor during message encoding was overcounted by 1 in most messages.

[RTI Issue ID MICRO-13129]

#### [Minor] OpenSSL PSK PSL implementation had NULL function pointers with no safeguards

The OpenSSL PSK PSL implementation had no safeguards against NULL function pointers when accessing any concrete transform method.

[RTI Issue ID MICRO-12721]

### 10.4.7 Content Filters and Query Conditions

#### [Major] Possible failure to deliver samples when prior samples were filtered

*Connext Micro* sometimes failed to deliver samples to a *DataReader* if the last processed sample was filtered, either by a configured ContentFilterQos or by a registered listener callback. This issue only occurred when a *DataWriter* stopped writing samples and a filtered sample was received out-of-order.

[RTI Issue ID MICRO-11917]

### 10.4.8 Logging

#### [Trivial] MAG logged Java version warning on Windows systems

Micro Application Generator (MAG) would incorrectly log the following warning when run on Windows systems:

"warning: Micro Application Generator has not been tested with Java . Unexpected behavior may occur."

[RTI Issue ID MAG-387]

### 10.4.9 APIs (C or Traditional C++)

#### [Major] Possible stack buffer overflow due to DGRAM interface name length

When the DGRAM interface was registered with a name longer than six characters (excluding the null terminator), *Connext Micro* produced a stack buffer overflow.

[RTI Issue ID MICRO-13176]

#### [Major] Setting large max sequence size would produce integer overflow

Calling `seq_set_maximum` with a very large value would overflow and result in a smaller-than-expected memory allocation.

[RTI Issue ID MICRO-12766]

#### [Major] DataWriter's _w_timestamp operations did not detect non-normalized timestamp inputs

*This issue was fixed in version 4.2.0, but not documented at that time.*

The `_w_timestamp` operations of a *DataWriter* did not detect non-normalized timestamp user inputs and they could cause undefined behavior. Providing a non-normalized timestamp value to these operations now results in the `BAD_PARAMETER` return code.

[RTI Issue ID MICRO-10626]

#### [Major] Undefined DDS_TIME_INVALID variables

`DDS_TIME_INVALID` and related variables were declared and documented, but not defined; when used, they caused linking errors.

[RTI Issue ID MICRO-6222]

#### [Minor] Misspelled OSAPI_LogProperty_INIITALIZER

The initializer for the `OSAPI_LogProperty_INIITALIZER` was misspelled and has been corrected to `OSAPI_LogProperty_INITIALIZER`. The misspelled macro is still available for backwards compatibility.

[RTI Issue ID MICRO-13453]

#### [Minor] DDS_Time_is_zero was not performing the correct evaluation

The public function DDS_Time_is_zero was not evaluating the DDS_Time in the correct way.

[RTI Issue ID MICRO-12798]

#### [Minor] Low resolution timestamps could cause Zero Copy samples to remain visible longer than expected

While using the Zero Copy v2 transport, if a sample was pushed out of a *DataWriter*'s history at the same time (with the same timestamp) as a *DataReader* was created, the *DataReader* would still be able to observe it. Additionally, Zero Copy samples with timestamps using a different clock than the *DataReader* could be missed.

[RTI Issue ID MICRO-10926]

#### [Minor] Initializing discovery property with NULL value incorrectly returned PRECONDITION_NOT_MET

Initializing the DPSE or DPDE discovery property with a `NULL` value incorrectly returned `PRECONDITION_NOT_MET` instead of `BAD_PARAMETER`.

[RTI Issue ID MICRO-10140]

#### [Minor] DDS_Entity_get_entity_kind was declared and documented but not defined

The function of the public API `DDS_Entity_get_entity_kind` was declared and documented but not defined. It could cause linking errors when used.

[RTI Issue ID MICRO-9226]

#### [Trivial] LatencyBudget was missing from the API documentation

LATENCY_BUDGET was missing from `DataReaderQos` or `DataWriterQos` in the C and C++ API reference.

[RTI Issue ID MICRO-6649]

#### [Trivial] False positive error messages when reattaching a condition to a WaitSet

If an error condition was attached to the same WaitSet more than once, *Connext Micro* would log a false-positive error message.

[RTI Issue ID MICRO-2857]

### 10.4.10 Generated Code (C, Traditional C++, and Modern C++)

#### [Major] Type registration API did not consistently report errors

The *DomainParticipant* `register_type` API should fail if a data type's maximum serialized size exceeds 4GB. However, code that was generated with `interpreted 0` would not cause the API to fail. Now, type registration fails consistently for types larger than 4 GB.

[RTI Issue ID MICRO-9889]

#### [Minor] Build error when generating an IDL that contained a long double on unsupported systems

When generating an IDL that contained a long double on systems that don't natively support long double, the resulting code would have an invalid variable.

[RTI Issue ID MICRO-3031]

#### [Minor] Superflous parent member generated in a derived C++ class when using -interpreted 0

When generating code using the rtiddsgen option `-interpreted 0` , a superflous parent member was generated in a derived C++ class.

[RTI Issue ID MICRO-2714]

#### [Minor] Generated AUTOSAR files placed in the wrong folder for AUTOSAR_CDD

*This issue was introduced in release 4.3.0 ER1 and did not impact release 4.2.0.*

When running rtiddsmag for the AUTOSAR_CDD application type, `Appgen.c`, `Appgen.h`, and `DdsCddType.arxml` were placed in the wrong folder if the application name was anything other than `MCU_DDS_CDD`.

[RTI Issue ID MAG-408]

#### [Trivial] Code Generator did not find Java when run from default directory

The *rtiddsgen* script could not locate the installed Java Runtime Environment (JRE) when run directly from the `bin/` directory.

[RTI Issue ID MICRO-11600]

### 10.4.11 Generated Code (Multiple Languages)

#### [Major] Invalid generated code when type was inside modules with duplicate names

*This issue was fixed in 4.0.0, but not documented at that time.*

*RTI Code Generator* would generate invalid code for C++ when it defined a type inside a set of modules and two of those modules had the same name. For example:

```
module A{
    module B{
        module A{
            struct myStruct {
                long m1;
            };
        };
    };
};
```

[RTI Issue ID MICRO-2416]

### 10.4.12 OMG Specification Compliance

#### [Minor] DDS_DURATION_INFINITE not compliant with OMG DDS specification

The value used for DDS_DURATION_INFINITE was not compliant with the OMG DDS specification.

[RTI Issue ID MICRO-10115]

### 10.4.13 Instances

#### [Major] Possible failure to send dispose/unregister messages when using async publication

When using asynchronous publication, *Connext Micro* may have failed to send dispose or unregister messages.

[RTI Issue ID MICRO-12789]

#### [Minor] Instance state transition inconsistent with Connext Professional

If a *DataWriter* called `dispose()` for an instance that was already unregistered, a *DataReader* would propagate the `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE` instance state even if the instance had not received any samples. This behavior was different from *Connext Professional*.

A new policy, DataReaderQos.protocol.propagate_dispose_of_unregistered_instances, has been added to control if the *DataReader* should propagate `DDS_NOT_ALIVE_DISPOSED_INSTANCE_STATE`, even when no samples have been received for the instance. When set to `false`, the behavior is identical to *Connext Professional*.

[RTI Issue ID MICRO-2346]

### 10.4.14 Crashes

#### [Major] Content-filtered DataWriter could crash or fail to send messages

When a *DataWriter* had writer-side content filtering enabled, calling `unregister_instance()` or `dispose()` with a valid instance handle and `NULL` instance data would cause a crash.

Calling either operation with non-`NULL` instance data that did not pass the content filter would instead cause messages to be silently discarded.

[RTI Issue ID MICRO-12610]

#### [Major] Potential crash from malformed data fragments

*Connext Micro* was not robust against malformed data fragments, which could lead to a crash.

[RTI Issue ID MICRO-12559]

#### [Minor] XML parsing errors with special characters in comments

Parsing failures occurred when XML files contained special characters (such as "º", Korean text, or other UTF-8 characters) in comments.

[RTI Issue ID MAG-397]

### 10.4.15 Hangs

#### [Critical] Potential deadlock when deleting a DomainParticipant that used the Flow Controller

When deleting a *DomainParticipant* that used the Flow Controller, a deadlock could occur; this was an unintended result of compiler optimization.

[RTI Issue ID MICRO-12547]

#### [Critical] Asynchronous DataWriter could silently stop sending samples and hang on deletion

When a *DataWriter* published large fragmented samples asynchronously through a DDS_FlowController, it could stop sending samples even though `write()` operations continued to succeed and heartbeats were still sent. Deleting the affected *DataWriter* or its *DomainParticipant* could then hang indefinitely.

[RTI Issue ID MICRO-13552]

#### [Major] DataWriters stalled indefinitely when sharing flow controller

When multiple *DataWriters* used the same DDS_FlowController with the same priority, only the first sample was sent successfully. After that sample was sent, the flow controller entered a state from which no further progress could be made, and all remaining *DataWriters* stalled indefinitely without sending any data.

[RTI Issue ID MICRO-2959]

### 10.4.16 Memory Leaks/Growth

#### [Major] Possible memory leaks in CDR_WstringSeq_set_maximum

If any of the buffers' allocation failed for a `CDR_Wstring` sequence, the previously allocated buffers would not be freed, causing a memory leak.

[RTI Issue ID MICRO-10036]

#### [Major] Possible memory leaks in CDR_WstringSeq_copy

If any of the buffers' allocation failed for a `CDR_Wstring` sequence, the previously allocated buffers would not be freed, causing a memory leak.

[RTI Issue ID MICRO-9969]

#### [Major] DDS_DomainParticipantFactory_get_instance() not memory safe on some architectures

On architectures with weak memory ordering (such as ARM), `DDS_DomainParticipantFactory_get_instance()` was not guaranteed to be memory safe.

[RTI Issue ID MICRO-8004]

#### [Major] Possible memory leak from failed write of a new instance

Resources may have been leaked if the value of `max_samples` was greater than `(max_instances * max_samples_per_instance)` and the *DataWriter*'s sample resources had been exhausted.

[RTI Issue ID MICRO-3129]

#### [Minor] Possible memory leak when finalizing Publishers or Subscribers with Partition QoS

Under specific conditions, *Publishers* and *Subscribers* that had user-defined Partition QoS policies would leak memory when finalized.

[RTI Issue ID MICRO-11336]

#### [Minor] Possible memory leak when Entities experienced errors on topic creation

During *DomainParticipant* topic creation, *Entities* would occasionally leak topic data memory if topic creation encountered an error.

[RTI Issue ID MICRO-11341]

### 10.4.17 Data Corruption

#### [Minor] Creating new DomainParticipant while receiving data could cause undefined behavior

Under rare conditions, creating a *DomainParticipant* while an existing *DomainParticipant* was processing an RTPS message could lead to undefined behavior.

[RTI Issue ID MICRO-10575]

#### [Minor] DDS_WstringSeq instance could be returned in an inconsistent state when set_maximum failed

If the function `DDS_WstringSeq_set_maximum()` failed and returned `RTI_FALSE`, the entire input sequence object could be returned in an inconsistent state.

[RTI Issue ID MICRO-9999]

### 10.4.18 Platform and Build Changes

#### [Major] Possible transmission failure when SocketOwner was configured

When *Connext Micro* was configured to use SocketOwner, the send socket was not bound to a local address before calling `TcpIp_UdpTransmit()`, which could cause the transmission to fail. The send socket is now bound to the configured `send_local_addr_id` with an OS-assigned ephemeral port (using `TCPIP_PORT_ANY`) before the first UDP transmission.

[RTI Issue ID MICRO-7848]

#### [Trivial] Not possible to disable thread semaphores

It was not possible to disable thread semaphores when building the full *Connext Micro* source code. Thread semaphores are used to suspend threads when using the `KEEP_ALL` HISTORY QoS policy, or when there are no available samples.

[RTI Issue ID MICRO-12557]

#### [Trivial] rtime-make would not compile Lightweight Security Plugin outside of source directory

If you called `rtime-make` outside of the *Connext Micro* source directory, the *Lightweight Security Plugin* source would not compile.

[RTI Issue ID MICRO-11192]

#### [Trivial] Excluing DDS flow control and fragmentation resulted in compile time error

The cmake option `RTIME_DDS_ENABLE_FLOW_CONRTOL` did not completely remote DDS flow control and fragmentation support from the core libraries and resulted in a compile time error.

[RTI Issue ID MICRO-2694]

### 10.4.19 Shipped Examples

#### [Minor] Cert example template used undesirable code constructs

The Cert example template code did not handle strings according to best practices.

[RTI Issue ID MICRO-10909]

#### [Trivial] The C++ example would attempt to delete invalid memory on specific failure conditions

The C++ generated example would attempt to delete invalid memory if the example failed before registering UDP.

[RTI Issue ID MICRO-3427]

### 10.4.20 Interoperability

#### [Critical] Failure to ignore vendor-specific PIDS

*Connext Micro* would not ignore vendor-specific parameter IDs (PIDs) when deserializing data from other DDS vendors.

[RTI Issue ID MICRO-12271]

### [Critical] NULL pointer access in AUTOSAR Classic when a ping message received from Connext Professional 7.3.0 or higher

An AUTOSAR Classic application using *Connext Micro* caused a NULL pointer access if a ping message was received from an application using *Connext Professional* 7.3.0 or higher.

[RTI Issue ID MICRO-7860]

### [Major] KeyFlag for fragmented data interpreted incorrectly

The *KeyFlag* (or K-bit) for fragmented data was interpreted from the wrong position and was always ignored.

[RTI Issue ID MICRO-11460]

### [Major] Samples of mutable types may have been dropped when bit FLAG_MUST_UNDERSTAND was set on key members

*Connext Micro* may have failed to deliver samples to applications, and reported them as lost, when receiving data on a *Topic* whose type:

- Had mutable extensibility;
- Derived from another type;
- Had at least one member marked as a key.

The failure would be triggered if the sending node set the bit FLAG_MUST_UNDERSTAND of the parameter IDs preceding each key member value in the serialized payload, as prescribed by the OMG 'Extensible and Dynamic Topic Types for DDS' specification.

This error may have only manifested in a cross-vendor interoperability scenario where *Connext* received data from a different vendor's DDS implementation. The error could also have been triggered if receiving data from a type with members marked as "must understand," for which the specification also requires the bit FLAG_MUST_UNDERSTAND be set in their serialized parameter IDs. Support for "must understand" members is introduced in this release. See Must-Understand Members in the *RTI Connext Core Libraries Extensible Types Guide* for more information.

*Connext Micro* will now correctly deliver all samples on *Topics* whose type may have triggered this error.

**Note:** To maximize interoperability with previous releases, *Connext Micro* does not set FLAG_MUST_UNDERSTAND for key members of mutable types by default. You can change this default and force compliance with the specification by using the Extensible Types Compliance Mask.

[RTI Issue ID MICRO-12758]

### 10.4.21 Vulnerabilities

The following vulnerabilities are fixed in this release.

See also RTI Connext Security Bulletins and Advisories for a complete list of vulnerabilities in RTI releases that have been published through the CVE® Program. That list may be more up-to-date.

#### [Critical] Potential stack buffer underflow in Connext Micro during message processing

##### User Impact without Security

A vulnerability in *Connext Micro* core libraries could have resulted in a stack buffer underflow when receiving malicious RTPS messages.

- CVSS v3.1 Base Score: 9.1 CRITICAL
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H
- CVSS v4.0 Base Score: 8.8 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:H/SC:N/SI:N/SA:N

##### User Impact with Security

There is no impact when enabling certain Security features; see Mitigations for more information.

##### Mitigations

- Enable *Lightweight Security Plugin*.
- Protect your network against unauthorized access.

[RTI Issue ID MICRO-12773]

#### [Critical] Potential out-of-bounds read in Connext Micro during message processing

##### User Impact without Security

A vulnerability in *Connext Micro* core libraries could have resulted in an out-of-bounds read when receiving malicious RTPS messages.

- CVSS v3.1 Base Score: 7.5 HIGH
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
- CVSS v4.0 Base Score: 8.8 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:N/VA:H/SC:N/SI:N/SA:N

#### User Impact with Security

There is no impact when enabling certain Security features; see Mitigations for more information.

#### Mitigations

- Enable *Lightweight Security Plugin*.
- Protect your network against unauthorized access.

[RTI Issue ID MICRO-12769]

#### [Minor] Potential-man-in-the-middle attack in Micro Application Generator when using Socket Appender in Apache Log4j™

The logging system in *Micro Application Generator* could be configured to employ the Socket Appender in Apache Log4j, which is affected by the vulnerability CVE-2025-68161.

This vulnerability has been fixed by upgrading Apache Log4j to version 2.25.3.

#### User Impact without Security

The associated CVE could be exploited by manipulating files in the *Connext Micro* installation.

- CVSS v3.1 Score: 4.0 MEDIUM
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N
- CVSS v4.0 Score: 2.1 LOW
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:H/AT:N/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:L/SA:N

#### User Impact with Security

Same impact as described in "User Impact without Security" above.

[RTI Issue ID MAG-247]

### 10.4.22 Other

#### [Critical] No communication on AUTOSAR systems with multiple IP addresses

*Connext Micro* would not receive any communication traffic in an AUTOSAR system when the following conditions were met:

- More than one IP address was configured for use in the AUTOSAR system.
- DDS communication used an IP address with an index greater than 0 (i.e., 1).

- *Connext Micro* used the default UDP interface factory properties, which set `enable_interface_bind` to `false`.

[RTI Issue ID MICRO-7783]

#### [Major] Corrected ARXML reference generation for basic types

Basic data types (defined as `SW-BASE-TYPE`) were incorrectly referenced as implementation data types when generating an ARXML from an XML file using *rtiarcgen*. References to the basic data types are now generated correctly.

[RTI Issue ID APIT-477]

#### [Major] DataReader could stop receiving samples due to resource miscount

A *DataReader* could incorrectly free a sample twice in specific conditions (e.g., when a RELIABLE sample was recommitted to a *DataReader*'s cache because the *DataReader* was unable to remove an older sample, e.g, due to a `KEEP_ALL` HISTORY QoS policy). This could prevent the *DataReader* from receiving samples from a matched *DataWriter*.

[RTI Issue ID MICRO-13068]

#### [Minor] Participant GUIDs generated on big-endian platforms may have reduced entropy

The GUID generation of the reference PSL for big-endian platforms may have reduced entropy and increased the likelihood of participants generating the same GUID.

[RTI Issue ID MICRO-13243]

#### [Minor] Errors in checksum documentation

The custom CRC example in the checksum documentation contained some incorrect structure names and type names.

[RTI Issue ID MICRO-13127]

#### [Minor] Incorrect max size calculation for primitive array with CDR_LONG_DOUBLE_TYPE

The padding associated with the maximum size was incorrectly calculated for primitive arrays with `CDR_LONG_DOUBLE_TYPE`.

[RTI Issue ID MICRO-13056]

#### [Minor] DataWriter silently stopped sending data when flow controller token budget was smaller thanfragment size

When a flow controller was configured such that `max_tokens * bytes_per_token` was less than `message_size_max`, the *DataWriter* could never accumulate enough tokens to send a single `DATA_FRAG` message and would silently stop sending data. When `bytes_per_token` is not unlimited, the fragment size used for fragmented messages is now the minimum of `bytes_per_token` and `message_size_max`, ensuring a single token is always sufficient to send one fragment.

[RTI Issue ID MICRO-2854]

#### [Trivial] Possible warning during DomainParticipant shutdown

*Connext Micro* may have printed the following benign warning during *DomainParticipant* shutdown:

```
WARNING: ModuleID=6 Errcode=39 X=1 E=0 T=1
"rtps"/RTPSInterface.c:5625/RTPS_Interface_unbind:
```

[RTI Issue ID MICRO-11135]

#### [Trivial] rtime-make and rtiddsgen script errors when running from a directory with a space in its name

If the path where the `rtime-make` or *rtiddsgen* script resides included a space, invoking it would cause a scripting error that included the following:

```
line 25: cd: too many arguments.
```

[RTI Issue ID MICRO-11099]

#### [Trivial] Formatting errors in help printout for rtime-make.bat

The `--help` printout for the batch file `rtime-make.bat` had incorrect syntax for `<name>` and `<target>` in the output.

[RTI Issue ID MICRO-2751]

## 10.5 Previous Releases

### 10.5.1 What's New in 4.2.0

*RTI Connext Micro* 4.2.0 is a feature release. See the Connext Releases page on the RTI website for more information on RTI's software release model.

The following features are new since *Connext Micro* 4.1.0.

#### Reduced core library size

This release separates the Extensible Types (XTypes) module from the core library, `rti_me`, into a separate library, `rti_me_ddsxtypes`. This change reduces the size of `rti_me`. The XTypes module is now optional and can be omitted if your application does not use its functionality.

Refer to the Migration Guide (if you have internet access) for more information on how this change affects existing *Connext Micro* applications.

#### Secure RTPS communication with the Lightweight Security Plugin

This release introduces the *Lightweight Security Plugin* for *Connext Micro*, a module that enables you to secure your application's RTPS messages with pre-shared key (PSK) encryption. The *Lightweight Security Plugin* includes a Platform Independent Library (PIL) and a Transform Platform Support Library (PSL) for interfacing with OpenSSL 3.5.

For more information on this feature, refer to the *Lightweight Security Plugin* section.

#### Optimize network bandwidth usage with content filtering

> **Attention:** This is an experimental feature in *Connext Micro* 4.2.0. It is not guaranteed to be consistent or supported and should not be used in production.
>
> Refer to *Experimental Features* for more information.

This release introduces content filtering to *Connext Micro*, enabling *DataReaders* to filter incoming sample data. This can reduce the network bandwidth usage in your application, since matching *DataWriters* can be configured to only send samples that pass the *DataReader*'s filter.

For more information on this feature, consult the *Content Filtering* section.

#### Improve system robustness with DomainParticipant rediscovery

This release adds the ability for a *DomainParticipant* with its `DomainParticipantQos.participant_name.name` set to a non-empty value to be rediscovered if it is restarted before its `lease_duration` expires. You can enable this feature by setting the `DomainParticipantQos.discovery.enable_participant_discovery_by_name` property to `DDS_BOOLEAN_TRUE`.

For more information on this feature, refer to the *DomainParticipant Discovery by Name* section.

#### Develop secure applications in XML with MAG

This release allows you to enable and configure the *Lightweight Security Plugin* while defining an application in XML format. Micro Application Generator (MAG) will then create the necessary code for *Connext Micro*.

For more details, refer to the *Lightweight Security Plugin* section of *Application Generation Using XML*.

#### Develop applications with content filtering in XML with MAG

This release allows you to enable and configure content filtering while defining an application in XML format. Micro Application Generator (MAG) will then create the necessary code for *Connext Micro*.

For more details, refer to the *Content filtering* section of *Application Generation Using XML*.

#### Enable DomainParticipant rediscovery in XML with MAG

This release allows you to enable *DomainParticipant* rediscovery while defining an application in XML format. Micro Application Generator (MAG) will then create the necessary code for *Connext Micro*.

For more information on this feature, see *DomainParticipant Discovery by Name*. You can enable *DomainParticipant* rediscovery via the `enable_participant_discovery_by_name` field in the PROPERTY QoS policy, as described in *Discovery Configuration*.

#### Enable endpoint discovery message queueing in XML with MAG

This release allows you to enable endpoint discovery message queueing while defining an application in XML format. Micro Application Generator (MAG) will then create the necessary code for *Connext Micro*.

This feature is included as a fix for a potential discovery issue; see *[Major] Possible mismatched endpoint messages during discovery* for more information. You can enable discovery message queueing via the `enable_endpoint_discovery_queue` field in the PROPERTY QoS policy, as described in *Discovery Configuration*.

#### Third-party and open source software changes

This release adds the following open source software as part of the *Lightweight Security Plugin*:

Table 10.5: Open Source Software Additions

| Software | Version |
|---|---|
| OpenSSL | 3.5.0 |

For more information, see the *Third-Party and Open Source Software* section.

### 10.5.2 What's Fixed in 4.2.0

The following are fixes since *Connext Micro* 4.1.0.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

#### Discovery

#### [Critical] Lost INFO_DST messages with GUIDPREFIX_UNKNOWN

*Connext Micro* may have silently discarded RTPS INFO_DST messages containing a *guidPrefix* of `0x0` (GUIDPREFIX_UNKNOWN).

[RTI Issue ID MICRO-8751]

#### [Major] DomainParticipants stopped sending announcements to initial peer locator after a discovered DomainParticipant went away

*This issue only affected patch release 4.1.0.1.*

If *DomainParticipant A* discovered *DomainParticipant B* on an initial peer locator, then lost *DomainParticipant B* and attempted to rediscover it, *DomainParticipant A* would stop sending discovery announcements to that initial peer locator.

[RTI Issue ID MICRO-10899]

#### [Major] Possible mismatched endpoint messages during discovery

If a *DomainParticipant* receives an endpoint discovery message and lacks sufficient resources to store the endpoint, *Connext Micro* will (by default) acknowledge the message as received, but discard the discovery information. This can lead to a mismatch in discovery states between *DomainParticipants*.

This release adds a new field, `DomainParticipantQos.discovery.enable_endpoint_discovery_queue`, and if this field is set to `DDS_BOOLEAN_TRUE`, endpoint discovery messages will instead be queued for later processing when resources become available.

---

**Note:** This new field assumes that there are sufficient resources available for discovery information and that the lack of resources is temporary (such as during a system restart). Setting this value to `DDS_BOOLEAN_TRUE` without sufficient resources may cause undefined behavior.

---

[RTI Issue ID MICRO-10044]

#### [Major] DomainParticipants continued to send DATA(p) messages to nonexistent remote participants

A *DomainParticipant* might have continued to send DATA(p) discovery messages to remote *DomainParticipants* that were once discovered but no longer exist.

[RTI Issue ID MICRO-8747]

#### [Major] DataReader failed to acknowledge samples when it received more instances than it could store

A *DataReader* failed to add key instances or acknowledge samples when it received more instances than its QoS supported.

[RTI Issue ID MICRO-8872]

#### [Minor] Participants using DPDE sent one extra announcement message

Participants using DPDE sent one more participant announcement message than expected for discovery purposes.

[RTI Issue ID MICRO-8257]

#### [Minor] Failed to delete a participant before sending at least one participant announcement

If `DomainParticipantFactory_delete_participant` was called before sending at least one participant announcement, `DomainParticipantFactory_delete_participant` would return a failure.

[RTI Issue ID MICRO-9828]

### Reliability Protocol and Wire Representation

#### [Major] Asynchronous flow controller may have stopped sending data

A flow controller configured with with an `added_bits_per_period` of `DDS_LENGTH_UNLIMITED` (such as the default flow controller) may have stopped sending data if the controller lost fragments while using reliable communication.

[RTI Issue ID MICRO-10991]

#### [Major] Reliable DataReader may have stopped receiving data

A reliable *DataReader* may have stopped receiving data if the data was received out of order from a *DataWriter* and the *DataWriter* was deleted before the *DataReader* could request the missing data.

[RTI Issue ID MICRO-10317]

### APIs (C or Traditional C++)

#### [Major] DDS_DomainParticipantFactory_finalize_instance failed if INTRA transport had been unregistered

The DDS_DomainParticipantFactory_finalize_instance function would fail if the INTRA transport had been unregistered previously in the test.

[RTI Issue ID MICRO-4481]

#### [Minor] Maximum blocking time is limited to 25 days

The maximum blocking time for APIs that supported blocking was approximately 25 days, even when the documentation stated a longer max blocking time.

[RTI Issue ID MICRO-9770]

### Generated Code (C, Traditional C++, and Modern C++)

#### [Major] Failed to serialize a union with only a default label

A union with only a default label could not be serialized. For example, the following union would fail to be serialized:

```
union Foo switch(int32) {
  default: long x;
};
```

[RTI Issue ID MICRO-9079]

### Hangs

#### [Critical] Application may have frozen during discovery of a remote participant

An application may have frozen during discovery of a remote participant when the built-in participant announcement *DataWriter* used asynchronous publishing. This was due to a race condition caused by the flow controller. Now, the built-in participant announcement *DataWriter* will only use synchronous publishing.

[RTI Issue ID MICRO-8901]

### Memory Leaks/Growth

#### [Minor] Memory leak in static discovery examples

The generated examples for C++ using static discovery leaked memory for the topic and type names.

[RTI Issue ID MICRO-10046]

#### [Major] Resources not freed when calling unregister_instance on a reliable Zero Copy DataWriter

When using a reliable Zero Copy *DataWriter*, calling `FooDataWriter_unregister_instance()` would not return the instance's resources to the appropriate pool for reuse.

[RTI Issue ID MICRO-10039]

### Platform and Build Changes

#### [Minor] Linker errors occurred when compiling HelloWorld examples for Windows systems

Compiling the HelloWorld examples for Windows systems in Release mode or with shared libraries resulted in linker errors. The build system will now correctly include the appropriate C runtime libraries and dynamic link libraries (DLLs) based on the Debug/Release configuration and whether static or dynamic linking is used.

[RTI Issue ID MICRO-8701]

### Shipped Examples

#### [Trivial] Failure to compile example generated for MAG

*This issue was fixed in 4.1.0, but not documented at that time.*

When generating an example for Micro Application Generator (MAG), two files (`<IDL>.xml` and `<IDL_Qos>.xml`) were not generated.

[RTI Issue ID MICRO-6801]

### Interoperability

#### [Major] Zero Copy communication failed when applications were started from different directories

When two applications were started from different working directories, they would fail to communicate when using the Zero Copy v2 transport.

---

**Note:** As a result of this fix, applications using Zero Copy transfer are not backwards compatible with previous versions of *Connext Micro*.

Refer to the Migration Guide for more information.

---

[RTI Issue ID MICRO-10391]

### Other

#### [Critical] Potential loss of queued samples when using asynchronous communication

If a sample was removed from a *DataWriter*'s history while the sample was queued for asynchronous publication, *Connext Micro* would not properly reacquire the network lock. This could result in losing queued samples while using `BEST_EFFORT` communication, or when resending samples with `RELIABLE` communication.

[RTI Issue ID MICRO-10947]

#### [Critical] Missed DATA_FRAG messages with multiple fragments

Under specific conditions, *Connext Micro* incorrectly processed RTPS messages that contained more than one DATA_FRAG submessage, resulting in missed messages.

[RTI Issue ID MICRO-9018]

#### [Major] Micro Application Generator (MAG) failed to start on macOS platforms

The `rtiddsmag` script failed to start on macOS platforms if `java` was not available in the `PATH` or `JAVA_HOME`. `rtiddsmag` will now use the JRE available with the host if `java` is not available in the `PATH` or `JAVA_HOME`.

[RTI Issue ID MAG-188]

#### [Major] DataWriter with finite timestamp failed to send data if date exceeded January 2038

A *DataWriter* configured with a finite source timestamp tolerance and DDS_BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS would fail to send data after 03:14:07 UTC 19 January 2038.

[RTI Issue ID MICRO-5858]

#### [Major] Connext Micro failed to parse malformed DATA_FRAG submessages

*Connext Micro* could enter an infinite loop when receiving a DATA_FRAG submessage with malformed inline QoS parameters.

[RTI Issue ID MICRO-10288]

#### [Major] Connext Micro would not work if year exceeded 2038

If the date was set beyond the year 2038, *Connext Micro* would not work. This was because the date is reported as a 32-bit unsigned integer; however, *Connext Micro* expects a signed 32-bit value and would therefore interpret the "wrap around" value as a negative number, causing an error.

[RTI Issue ID MICRO-2295]

#### [Minor] Multiple endpoints did not match properly when using Zero Copy transfer

When using the Zero Copy v2 transport, multiple endpoints would not match properly if both *DomainParticipants* were enabled before creating *DataReaders* or *DataWriters.*

[RTI Issue ID MICRO-10162]

#### [Minor] SYSTEM_RESOURCE_LIMITS.max_components could not be changed

*Connext Micro* applications could not change the `SYSTEM_RESOURCE_LIMITS.max_components`.

[RTI Issue ID MICRO-3571]

#### [Trivial] rtime-make did not support building multiple targets on Windows systems and failed to report some CMake errors

The rtime-make batch script would fail to build additional targets after the first `--` parameter in the command line on Windows systems. Additionally, the rtime-make batch and shell scripts failed to report some CMake errors correctly.

[RTI Issue ID MICRO-10913]

### 10.5.3 What's New in 4.1.0

*RTI Connext Micro* 4.1.0 is a feature release. See the Connext Releases page on the RTI website for more information on RTI's software release model.

The following features are new since *Connext Micro* 4.0.1.

#### Platform-independent code is now separate from OS and network stack integration

This release includes precompiled *Connext Micro* binaries in two formats:

- Platform Independent Libraries (PIL): binaries that support the basic features of *Connext Micro*.
- Platform Support Libraries (PSL): binaries that support OS and network stack integration.

This split allows for different platform-specific PSLs to be written for the same PIL without needing to recompile the *Connext Micro* code. Previous releases of *Connext Micro* were delivered as integrated libraries with the OS and network stack code included, which could not be changed without recompiling the entire library. See the *Library descriptions* section for more information on this change.

Since these split libraries are provided precompiled, *Connext Micro* can be used out-of-the-box without building the source code first; see *Getting Started* and *Developing Applications*. However, RTI also provides the source code for the PSL with supported architectures. Refer to *Building the PSL* for instructions on how to build your own PSL from source.

**Warning:** Split libraries **do not** automatically register the UDP transport. This is because *Connext Micro* makes no assumptions about which transports are available when using split libraries. In order to use the UDP transport included in the PSL, add the following code before creating a *DomainParticipant*:

```
/* Register the UDP transport */
struct UDP_InterfaceFactoryProperty *udp_property;

/* allocate and set udp_property */

RT_Registry_register(registry,NETIO_DEFAULT_UDP_NAME,
         UDP_InterfaceFactory_get_interface(),
         (struct RT_ComponentFactoryProperty*)udp_property,NULL),

DDS_StringSeq_set_maximum(&participant_qos.transports.enabled_transports,1);
DDS_StringSeq_set_length(&participant_qos.transports.enabled_transports,1);
*DDS_StringSeq_get_reference(&participant_qos.transports.enabled_transports,0) = DDS_
↪String_dup("_udp");

DDS_StringSeq_set_maximum(&participant_qos.discovery.enabled_transports,1);
DDS_StringSeq_set_length(&participant_qos.discovery.enabled_transports,1);
*DDS_StringSeq_get_reference(&participant_qos.discovery.enabled_transports,0) = DDS_
↪String_dup("_udp://");

DDS_StringSeq_set_maximum(&participant_qos.user_traffic.enabled_transports,1);
DDS_StringSeq_set_length(&participant_qos.user_traffic.enabled_transports,1);
*DDS_StringSeq_get_reference(&participant_qos.user_traffic.enabled_transports,0)
= DDS_String_dup("_udp://");
```

### Transfer large data samples quickly with Zero Copy v2

This release adds a new transport, Zero Copy v2, which can perform Zero Copy data transfer. Zero Copy transfer allows you to move large data samples without copying them, which increases throughput and reduces latency.

You can now set up Zero Copy transfer to use either the Shared Memory Transport (which was available in previous versions of *Connext Micro*) or the new Zero Copy v2 transport. The main difference between the two transports is that the Shared Memory Transport is interoperable with *Connext Professional* and the Zero Copy v2 transport is interoperable with select versions of *Connext Cert*.

For more details, refer to *Zero Copy Transfer* and *Zero Copy v2 Transport*.

### Enable and configure Zero Copy transfer with MAG

This release allows you to enable and configure the *Zero Copy v2 Transport* while defining an application in XML format. Micro Application Generator (MAG) will then create the necessary code to enable and configure the Zero Copy transport in *Connext Micro*.

For details on how to enable Zero Copy v2 with MAG, refer to *MAG Command-Line Options* and *Transport and Discovery Configuration*.

#### Enhance data reliability by detecting and discarding corrupted RTPS messages

This release adds support for detecting and discarding corrupted RTPS messages. This improves data reliability and provides basic security by ensuring that the data has not been modified in transit.

A Cyclic Redundancy Check (CRC) is computed over the DDS RTPS message (including the RTPS Header), which is sent as a new RTPS submessage. The subscribing application can detect this new submessage and validate the contained CRC. Optionally, when a corrupted RTPS message is detected, the message can be dropped.

To enable the use of CRC in a *DomainParticipant*, there are three new fields in the WireProtocolQoSPolicy:

- `compute_crc`: when enabled at the sending application, sends the CRC.
- `check_crc`: when enabled, drops corrupted messages.
- `require_crc`: when enabled, ignores participants with `compute_crc` set to false.

Refer to *Message Integrity Checking* for details.

#### Develop more reliable applications with MAG

This release adds support to Micro Application Generator (MAG) for the Cyclic Redundancy Check feature (see *Enhance data reliability by detecting and discarding corrupted RTPS messages*). This allows you to develop applications with MAG that have improved data reliability and basic security.

MAG now supports the following fields in the WireProtocolQosPolicy:

- `compute_crc`
- `check_crc`
- `require_crc`
- `computed_crc_kind`
- `allowed_crc_mask`

Refer to *Message Integrity Checking* for details.

#### Guarantee compatibility with Connext Professional with MAG when using the Shared Memory Transport

This release adds support to Micro Application Generator (MAG) for the `dds.transport.minimum_compatibility_version` property, which you can set via the PROPERTY QoS policy for the *DomainParticipant*.

`dds.transport.minimum_compatibility_version` changes the value of the new field pro_minimum_compatibility_version that has been added to the shared memory interface factory property. This property sets the minimum version of *Connext Professional* to be compatible with when using shared memory.

The default value for this field is `DDS_PRODUCTVERSION_UNKNOWN`.

Refer to *SHMEM Configuration* for details.

#### Improve control of data distribution to multicast addresses with new UDP transport options

This release adds the following new options to the UDP transport to further control how *Connext Micro* sends data to multicast addresses:

- disable_multicast_bind: controls whether *Connext Micro* will bind to a multicast address receive address (if set to 0) or bind to ANY multicast address (if set to 1).
- multicast_loopback_disable: controls whether *Connext Micro* puts multicast packets onto the loopback interface.
- disable_multicast_interface_select: controls whether *Connext Micro* will use `multicast_interface` or `allow_interface/deny_interface` to select the interface for sending to multicast addresses.

Refer to *UDP Transport* for more information on these options.

#### Develop applications with new UDP transport options with MAG

This release adds support to Micro Application Generator (MAG) for the following new fields when configuring the UDP transport:

- disable_multicast_bind
- multicast_loopback_disable
- disable_multicast_interface_select

Refer to *UDP Configuration* for more information on these properties.

#### Build Connext Micro libraries conveniently with symlinks

This release adds support for using the `cmake --install` command with symbolic links (symlinks) as well as full paths.

Refer to *Building Connext Micro for Common Platforms* for more information on how to use CMake build commands.

### 10.5.4 What's Fixed in 4.1.0

The following are fixes since *Connext Micro* 4.0.1.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

#### Discovery

##### [Major] Participants did not perform discovery correctly when on_data_on_readers callback was set

When the `on_data_on_readers` callback was enabled on participants, the participants would not perform discovery correctly.

[RTI Issue ID MICRO-7412]

##### [Minor] Incorrect data fragmentation of discovery messages for DomainParticipant built-in topic

*Connext Micro* incorrectly fragmented *DomainParticipant* discovery messages if the Maximum Transmission Unit (MTU) was set to a value lower than the *DomainParticipant* built-in data. Fragmentation of the *DomainParticipant* built-in topic is not supported. The system now ensures that the creation of a *DomainParticipant* will fail if the minimum MTU across all transports used for discovery is less than the size of the *DomainParticipant* discovery message.

[RTI Issue ID MICRO-7545]

#### Usability

##### [Minor] MAG failed to generate warnings for certain unsupported QoS

*Micro Application Generator* (MAG) did not generate warnings for certain unsupported QoS values in XML configurations. This could lead to runtime errors when creating participants with incorrect configurations.

[RTI Issue ID MAG-176]

### Transports

#### [Major] Multicast sockets always bound to the multicast address on non-Windows platforms

*Connext Micro* always bound multicast receive sockets to the multicast address upon creation, even if enable_interface_bind had not been set. This only occurred on non-Windows platforms.

Now, disable_multicast_bind can optionally disable binding a multicast receive socket to its multicast receive address.

See *Improve control of data distribution to multicast addresses with new UDP transport options* for more information on disable_multicast_bind.

---

**Note:** On some platforms, you may need to set disable_multicast_bind to 1 to interoperate *Connext Micro* with *Connext Professional* because *Connext Professional* does not bind multicast receive sockets to the multicast address.

---

[RTI Issue ID MICRO-8451]

### Reliability Protocol and Wire Representation

#### [Critical] Failed to deliver samples when prior samples were lost

*Connext Micro* would sometimes fail to deliver samples to a *DataReader* if the last processed sample was a HEARTBEAT message which indicated lost samples.

[RTI Issue ID MICRO-7317]

#### [Major] Delayed or failed to deliver large samples on unreliable networks

The RELIABILITY protocol may have struggled to repair lost data fragments. In unreliable network conditions, fragmented samples of large data could be delayed or fail to deliver at all if the samples were written faster than they could be repaired. Now, the RELIABILITY protocol for lost fragments has been improved so that fragmented samples are repaired quickly on an unreliable network.

[RTI Issue ID MICRO-8255]

#### [Minor] Piggyback heartbeats were not included when using asynchronous publication

*DataWriters* did not include piggyback heartbeats with non-fragmented samples when using asynchronous publication. They are now included at the rate configured by DDS_RtpsReliableWriterProtocol_t::heartbeats_per_max_samples, regardless of the publication mode.

[RTI Issue ID MICRO-8173]

### APIs (C or Traditional C++)

#### [Major] DDS_DomainParticipantFactory_delete_participant was not thread-safe

DDS_DomainParticipantFactory_delete_participant was not thread-safe. A race condition could occur if multiple *DomainParticipants* were deleted at the same time in the same process space.

[RTI Issue ID MICRO-8055]

#### [Minor] C++ constructor did not allocate memory for Topic and Type names

The C++ constructor for DDS_PublicationBuiltinTopicData and DDS_SubscriptionBuiltinTopicData did not allocate memory for the **topic_name** and **type_name** attributes.

Please refer to the API Reference for more information on these APIs.

[RTI Issue ID MICRO-7110]

#### [Trivial] C++ API Reference contained incorrect «cert» references

The C++ API Reference contained `<<cert>>` references, but *Connext Cert* does not support the C++ API.

[RTI Issue MICRO-3216]

#### [Trivial] Non-descriptive API failure messages

The following APIs have updated failure messages:

- FooDataWriter_get_loan failed with `UNSUPPORTED` if your type was annotated with `@transfer_mode(SHMEM_REF)` and no transports that support Zero Copy transfer were enabled. It now fails with `PRECONDITION_NOT_MET`.
- FooDataWriter_discard_loan failed with `UNSUPPORTED` if your type was annotated with `@transfer_mode(SHMEM_REF)` and no transports that support Zero Copy transfer were enabled. It now fails with `PRECONDITION_NOT_MET`.
- FooDataWriter_get_loan failed with `ERROR` if all samples had been loaned to your application. It now fails with `OUT_OF_RESOURCES`.

[RTI Issue ID MICRO-7929]

### XML Configuration

#### [Minor] Invalid code when using a flow controller

*Micro Application Generator* (MAG) generated some invalid code when using a flow controller in the following scenarios:

- Trying to use one of the built-in flow controllers.
- `SYNCHRONOUS_PUBLISH_MODE_QOS` was configured.

[RTI Issue ID MAG-174]

### Crashes

#### [Critical] Segmentation fault occurred when an asynchronous publisher tried to send a missing data fragment

When an asynchronous publisher tried to send a data fragment that was no longer available, a segmentation fault occurred.

[RTI Issue ID MICRO-7982]

#### [Critical] Segmentation fault occurred when creating or finalizing a DataWriter or DataReader while using asynchronous publication

A segmentation fault could have occurred when creating or finalizing a *DataWriter* or *DataReader* if the built-in writers were using asynchronous publication. The built-in writers would automatically use asynchronous publication if the `max_message_size` property of the UDP transport was set to a low value (~1400). This issue only occurred if the *DataWriter* or *DataReader* was created or finalized while the *DomainParticipant* was matched with another *DomainParticipant*.

[RTI Issue ID MICRO-8111]

#### [Critical] Potential race condition and crash occurred when a DataWriter waited for resources

A race condition could occur when a *DataWriter* waited for resources to be freed while sending data asynchronously. This could cause a crash, since the *DataWriter* state was incorrectly updated.

[RTI Issue ID MICRO-8001]

#### [Critical] Potential race condition and crash occurred when a DataWriter unmatched with a remote entity

A race condition could occur when a *DataWriter* using asynchronous publishing unmatched with a remote entity while data was being sent to the same entity. This could cause a crash.

[RTI Issue ID MICRO-8170]

#### [Minor] Integer overflow when setting MTU lower than 448 bytes

*Connext Micro* could produce errors if the Maximum Transmission Unit (MTU) for any transport was set lower than 448 bytes. *Connext Micro* now ensures that the MTU across all enabled transports is greater than 448 bytes. If the MTU requirement is not met, entity creation will fail.

[RTI Issue ID MICRO-7530]

### Hangs

#### [Critical] Ungracefully terminated QNX processes using SHMEM transport prevented startup of new processes due to unclosed POSIX semaphores

If a QNX application using the shared-memory transport was ungracefully shut down, crashed, or otherwise had an abnormal termination while holding a POSIX semaphore used by the transport (for example, while sending data through the shared-memory transport), *Connext* applications launched after that point on the same domain may have waited forever for that semaphore to be released.

Now on QNX 7.1 and greater, the usage of POSIX semaphores has been replaced with robust pthread mutexes. Abnormal termination of an application while holding a mutex will no longer result in a *Connext* application launched after that point hanging.

As a result, *Connext Micro* 4.1.0 will **not** be backward compatible with previous versions of *Connext Micro* when using the shared memory transport on QNX 7.1 and greater. *Connext Micro* 4.1.0 will now be compatible with *Connext Professional* 7.3.0 on QNX 7.1 and greater.

[RTI Issue ID MICRO-6013]

### Memory Leaks/Growth

#### [Minor] Failed to cleanup resources when DomainParticipant creation failed

If the creation of a *DomainParticipant* failed, the allocated resources were not correctly cleaned up.

[RTI Issue ID MICRO-6500]

#### Data Corruption

##### [Critical] Potential data corruption from race condition when using KEEP_LAST and publishing asynchronously

Data samples may have been corrupted if both of the following were true:

- A race condition occurred while sending fragmented samples.
- The fragmented samples were sent while a `HISTORY.kind` of DDS_KEEP_LAST_HISTORY_QOS caused samples to be removed and reused.

[RTI Issue ID MICRO-8314]

#### Interoperability

##### [Critical] Incorrect handling of RTPS messages with submessages from different participants

When an RTPS message that contained submessages from multiple participants was received, *Connext Micro* incorrectly treated each submessage as though it was from the participant whose GUID prefix was in the RTPS header. *Connext Micro* does not send RTPS messages with submessages from different participants, but other DDS vendors may do this, which would have led to various communication issues and a lack of interoperability.

[RTI Issue ID MICRO-5984]

#### Other

##### [Critical] Lost samples or fragments not repaired when using a flow controller

When using a flow controller, repair packets may not have been sent by a *DataWriter* to a *DataReader* to replace lost fragments or samples. The writer would only send repair packets when a new sample was written, resulting in repairs never being sent if another sample was not written. Repair packets will now be sent by the flow controller as soon as they are required.

[RTI Issue ID MICRO-7403]

##### [Major] Enabling asynchronous publication on DataWriter caused RTPS messages to fail

If the `publish_mode.kind` of a *DataWriter*'s QoS was configured to be `ASYNCHRONOUS_PUBLISH_MODE_QOS`, RTPS messages from that *DataWriter* may have failed to send. This condition occurred specifically if asynchronous publication was enabled AND the sample type did not require RTPS fragmentation.

[RTI Issue ID MICRO-7219]

#### [Major] Finalizing a participant might have failed when using DPSE

`DDS_DomainParticipant_finalize()` may have failed if the participant was using the DPSE discovery plugin. This issue was most likely to occur on macOS® platforms.

[RTI Issue ID MICRO-7870]

#### [Major] Samples with meta-information were not delivered to the user if they arrived when history cache was full

When a *DataReader*'s history cache was full, samples containing meta-information from matched *DataWriters* were not delivered to the user.

[RTI Issue ID MICRO-8063]

#### [Minor] Flow controllers incorrectly delayed sending packets

The token-bucket flow controller may have waited to send packets until its next period, even if more tokens were available during the current period.

[RTI Issue ID MICRO-8024]

#### [Minor] DataWriters with KEEP_ALL History may not have sent all historical samples

*DataWriters* with History kind set to `DDS_KEEP_ALL_HISTORY_QOS` and Durability kind set to `DDS_TRANSIENT_LOCAL_DURABILITY_QOS` would send historical samples only up to the History depth. *DataWriters* now send historical samples up to max_samples_per_instance.

[RTI Issue ID MICRO-8505]

#### [Trivial] Illegal reflective access warning when running MAG with OpenJDK™ 11

*This issue was fixed in 4.0.1, but not documented at that time.*

Running Micro Application Generator (MAG) with OpenJDK 11 generated the following warning:

```
WARNING: An illegal reflective access operation has occurred

WARNING: Illegal reflective access by com.rti.micro.appgen.utils.QosUtils (file:/.../rti_
↪connext_dds_micro-4.0.0/rtiddsmag/class/rtiddsmag.jar) to field java.lang.String.value

WARNING: Please consider reporting this to the maintainers of com.rti.micro.appgen.utils.
↪QosUtils

WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective␣
↪access operations
```

(continues on next page)

(continued from previous page)

```
WARNING: All illegal access operations will be denied in a future release
```

MAG has been updated to use OpenJDK 17, which does not generate this warning.

[RTI Issue ID MAG-172]

#### [Trivial] Incorrect example instructions in the User's Manual

*Examining the application* in the User's Manual stated incorrectly that the files for the example were included with *Connext Micro*. The instructions have been updated to reflect that the files are generated with `rtiddsmag`.

[RTI Issue ID MICRO-7376]

#### [Trivial] Micro transformation example failed to compile

*Connext Micro* failed to compile with the Visual Studio® 2010 solution files included in the transformation example. The build process has been updated to compile with a new CMake file.

[RTI Issue ID MICRO-7171]

### 10.5.5 What's New in 4.0.1

*RTI Connext Micro* 4.0.1 is an Early Access Release, based on release 4.0.0.

The following features are new since *Connext Micro* 4.0.0.

#### Enable or disable padding bits with PROPERTY QoS policy in DomainParticipant and DataWriter

Micro Application Generator (MAG) adds support for enabling and disabling sending padding bits. This feature is part of a fix for a data corruption issue; see *[Critical] DataReader on a Topic with appendable type could receive samples with incorrect value* for more information.

Padding bits can be set with the following property, via the PROPERTY QoS policy:

- dds.xtypes.compliance_mask, to enable or disable padding bits for the *DomainParticipant* or *DataWriter*. The only valid values supported by MAG for this property are 0 and 0x00000008. MAG will report an error if a different value is set.

---

**Note:** In previous releases, MAG ignored PROPERTY QoS values, but now it parses all PROPERTY QoS values configured in XML and adds those values when generating the code. However, it ignores every PROPERTY QoS property that is not **dds.xtypes.compliance_mask**. Future *Connext Micro* releases may add support for additional properties.

---

#### Generate examples with new template options for Code Generator

In this release, some examples that were previously included in a *Connext Micro* installation have been removed. Instead, examples can be generated from templates included with the *RTI Code Generator*.

This release introduces two new *Code Generator* command-line options, `-showTemplates` and `-exampleTemplates`.

The `-showTemplates` option prints and generates an XML file containing a list of available example templates in your *Connext Micro* installation, organized per language.

The `-exampleTemplate` option generates an example you specify, instead of the default one.

When you use the `-exampleTemplate` option, you can specify one of the example templates in `$RTIMEHOME/rtiddsgen/resource/templates/example/<language>/<templateName>/`. You may also create your own templates and place them in this directory.

You must also use one of the following command-line options:

- `-create examplefiles`
- `-update examplefiles`
- `-example`

*Code Generator* will then generate the example you specified. For example:

```
rtiddsgen -language C++ -example -exampleTemplate <exampleTemplateName> foo.idl
```

For more information, please refer to *Example Applications*.

### 10.5.6 What's Fixed in 4.0.1

The following are fixes since *Connext Micro* 4.0.0.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

#### Discovery

#### [Trivial] Possible error message during discovery with Connext Professional

The following log message may have been printed during discovery with a *Connext Professional* application:

```
[1712165262.572102999]ERROR: ModuleID=4 Errcode=27 X=1 E=0 T=1 netio/NETIOPacket.c:87/
↪NETIO_Packet_set_head: delta=20
```

This message was benign and did not indicate any failures.

[RTI Issue ID MICRO-6594]

#### Usability

#### [Minor] rtiddsgen failed to run if the default shell was not bash compatible

If the default shell on a macOS or Linux system was not bash (e.g., tcsh), *rtiddsgen* would fail to execute.

[RTI Issue ID MICRO-6539]

#### [Trivial] Self toolchain file missing from the Connext Micro bundle

The `self.tc` toolchain file referred to in the documentation was missing from the *Connext Micro* bundle.

[RTI Issue ID MICRO-6536]

#### [Trivial] Empty README.txt generated for an example

When generating an example using the `-example` option for *rtiddsgen*, the generated README.txt file was empty.

[RTI Issue ID MICRO-6642]

#### APIs (C or Traditional C++)

#### [Minor] Unexpected behavior when copying a DDS_UnsignedShortSeq with 0 length

When copying a `DDS_UnsignedShortSeq` with 0 length, the destination sequence length was not set to 0.

[RTI Issue ID MICRO-2756]

#### [Minor] Missing C++ APIs for discovery operations

The following functions were missing from the C++ API:

- get_discovered_participants
- get_discovered_participant_data
- get_matched_subscriptions
- get_matched_subscription_data

- get_matched_publications
- get_matched_publication_data

For more information on these functions, please refer to the C++ API Reference.

[RTI Issue ID MICRO-6462]

#### [Trivial] C++ examples used the undocumented get_reference API

The C++ examples used the undocumented `get_reference` API. C++ examples now use the `[]` operator.

[RTI Issue ID MICRO-3104]

### Generated Code (C, Traditional C++, and Modern C++)

#### [Major] Incorrect generated code when using IDL whose name starts with a number

The generated code for an IDL whose name started with a number was incorrect and did not compile. The generated code contained some ifdef instructions that started with a number, which was not valid because an identifier must start with a letter (or underscore).

Now, invalid identifier characters are converted to '_' in the ifdef instruction.

[RTI Issue ID MICRO-2066]

#### [Major] Code generated for a FLAT_DATA type failed to compile when using namespace option

Code generated for a FLAT_DATA type failed to compile when using the `-namespace` option to run *rtiddsgen*.

[RTI Issue ID MICRO-6788]

#### [Major] Code generated for an aliased sequence of an aliased string failed to compile

The code generated for an aliased sequence of an aliased string failed to compile. For example, the following IDL would fail:

```
typedef string<2> MyString10;
typedef sequence<MyString10,4> MyStringSeq10;

struct SequenceType4 {
   string<64> msg;
   MyString msg2;
   MyStringSeq10 seq;
};
```

[RTI Issue ID MICRO-6824]

### Crashes

#### [Minor] Potential segmentation fault while creating entities

A segmentation fault could occur while creating certain entities if *Connext Micro* ran out of memory. *Connext Micro* will now detect this condition and return an error.

This issue only affected non-CERT profiles.

[RTI Issue ID MICRO-3396]

### Data Corruption

#### [Critical] DataReader on a Topic with appendable type could receive samples with incorrect value

A *DataReader* subscribing to a *Topic* on an appendable type may have received incorrect samples from a matching *DataWriter*.

The problem only occurred when the *DataWriter* published a type with fewer members than the *DataReader* type. For example, consider a *DataWriter* on FooBase and a *DataReader* on FooDerived:

```
@appendable struct FooBase {
   sequence<uint8,1024>base_value;
};

@appendable struct FooDerived {
   sequence<uint8,1024> base_value;
   @default(12) uint8 derived_value;
};
```

When the *DataWriter* published a sample with type FooBase, in some cases the *DataReader* received a sample in which the field `derived_value` was set to 0 instead of 12.

This issue was caused by a bug in which *Connext* did not set the padding bits in the encapsulation header for a serialized sample as required by the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3. As a result, some of the padding bytes were interpreted as data.

---

**Note:** This fix may lead to a compatibility issue causing a *Connext Micro DataWriter* to not match with a *Connext Micro* or *Connext Cert DataReader*.

For more information, see Extensible Types Compliance Mask in the *Core Libraries Extensible Types Guide* if you have Internet access.

Padding bits can be disabled with the dds.xtypes.compliance_mask property for backwards compatibility with the following releases:

- *Connext Micro* 2.4.12 and earlier
- *Connext Micro* 2.4.13.2-5
- *Connext Micro* 2.4.14 and 2.4.14.1
- *Connext Cert* 2.4.12.1
- *Connext Cert* 2.4.13.1
- *Connext Cert* 2.4.15.1
- *Connext Micro* 3
- *Connext Micro* 4.0.0

[RTI Issue ID MICRO-5930]

#### [Critical] Undefined behavior using XCDR2 with keyed topic types with key union members

Using XCDR encoding version 2 (XCDR2) with keyed topic types with key union members was not supported. For example:

```
union MyUnion switch(long) {
   case 0:
       long m_long;
   case 1:
       short m_short;
};

struct StructWithUnionKey {
   @key MyUnion m_union;
   long m_long;
};
```

The behavior was undefined if any of your topic types had a union key member. The results varied, from a potential segmentation fault to an incorrect key hash in which two instances were considered equal.

[RTI Issue ID MICRO-5933]

#### [Critical] Incorrect keyhash generated when receiving data without a keyhash from a node with different endianness

A *DataReader* would generate an incorrect keyhash for a received sample if all of the following were true:

- The *DataReader* did not receive a keyhash in a sample for a keyed type.
- The *DataReader* used the FLAT_DATA language binding.
- The sample was sent from a node with different endianess than the *DataReader*.

Also, a *DataReader* would generate an incorrect keyhash for a DISPOSE or UNREGISTER sample if:

- The *DataReader* did not receive a keyhash in a DISPOSE or UNREGISTER sample for a keyed type.
- The *DataReader* used IDL compiled with `-interpreted 1` (the default).
- The DISPOSE or UNREGISTER sample was sent from a node with a different endianess than the *DataReader*.

---

**Note:** Both *Connext Micro* and *Connext Professional* send a keyhash by default, but *Connext Professional* can be configured to send without a keyhash.

---

[RTI Issue ID MICRO-6870]

#### Interoperability

#### [Major] Incorrect deserialization of CDR encapsulation padding bit

The number of padding bytes in a sample were de-serialized incorrectly. This resulted in samples being dropped if the number of padding bytes was not zero.

[RTI Issue ID MICRO-6799]

### 10.5.7 What's New in 4.0.0

*RTI Connext Micro* 4.0.0 is an Engineering Release, based on release 3.0.3.

The following features are new since *Connext Micro* 3.0.3.

#### Enhanced performance for asynchronous DataWriters

This release reduces the contention between *DataWriters* and the asynchronous publication thread (used for flow-control of samples and publishing fragmented samples). Previously, *DataWriters* would block while the asynchronous publication thread was sending data. In this release, the asynchronous publication thread uses a separate critical section from the *DataWriter*'s write API, which allows the *DataWriter* to write samples while the asynchronous publication thread is sending data.

Please note the following limitations:

- It is not possible to send and receive data at the same time.
- The asynchronous publication thread and the *DataWriter* will contend for the same critical section when the asynchronous publication thread starts or finishes sending a sample. This is because the *DataWriter* is loaning samples to the asynchronous publication thread instead of copying them, and the ownership transfer of samples from the *DataWriter* to the asynchronous

publication thread (and from the asynchronous publication thread to the *DataWriter*) is protected.

In addition, the User's Manual has not been updated for this release; some sections do not reflect the impact of these changes. Specifically, note the following:

- Each *DataWriter* allocates 1 additional mutex.
- Each *DomainParticipant* allocates 2 additional mutexes, plus 1 mutex per flow-controller (3 by default).
- Each *DataWriter* allocates an additional (`max_routes_per_reader` * `max_fragmented_samples` * `max_remote_readers` * 464) bytes. Future releases may reduce this.

#### Further control which entities communicate with each other using new Partition QoS policy

The PARTITION QoS policy provides a method to prevent *Entities* that have otherwise compatible QoS policies from matching—and thus communicating with—each other. Much in the same way that only applications within the same DDS domain will communicate with each other, only *Entities* that belong to the same partition can talk to each other.

See information on *Partitions* in the User's Manual chapter for more information.

#### Store additional entity-related information that is passed between applications during discovery using new User/Topic/Group Data QoS policies

*Connext Micro* now provides areas where your application can store additional information related to DDS *Entities*. How this information is used is up to user code. *Connext Micro* distributes this information to other applications as part of the discovery process; however, *Connext Micro* does not interpret the information. Use cases are usually application-to-application identification, authentication, authorization, and encryption.

There are three User Discovery Data QoS policies:

- USER_DATA: associated with *DomainParticipants*, *DataWriters*, and *DataReaders*.
- TOPIC_DATA: associated with *Topics*.
- GROUP_DATA: associated with *Publishers* and *Subscribers*.

See information on *User Discovery Data* in the User's Manual chapter for more information.

#### Verify that locally created participant GUIDs are unique within a DomainParticipantFactory

When a *DomainParticipant* is created, *Connext Micro* now checks that the GUID is not already in use by another *DomainParticipant* created from the same *DomainParticipantFactory*.

### Micro Application Generator (MAG)

#### Support for Partition QoS policy in MAG

Micro Application Generator (MAG) now supports the PARTITION QoS policy. Instead of ignoring the Partition QoS values, as it did in previous releases, MAG now parses the values configured in XML and adds those values when generating the code.

The following partition-related *DomainParticipant* QoS resource limits are also now supported:

- **max_partitions**
- **max_partition_cumulative_characters**
- **max_partition_string_size**
- **max_partition_string_allocation**

See the *Partitions* chapter in the User's Manual for more information on this QoS policy.

#### Support for GROUP_DATA, USER_DATA, and TOPIC_DATA QoS policies in MAG

Micro Application Generator (MAG) now supports the GROUP_DATA, USER_DATA, and TOPIC_DATA QoS policies. Instead of ignoring these QoS values, as it did in previous releases, MAG now parses the values configured in XML and adds those values when generating the code.

MAG also supports the group_data, user_data, and topic_data elements:

- **user_data** in the *DomainParticipant*, *DataWriter*, and *DataReader* QoS
- **topic_data** in the *Topic* QoS
- **group_data** in the *Publisher* and *Subscriber* QoS
- The following *DomainParticipant* QoS resource limits:
  - **participant_user_data_max_length**
  - **participant_user_data_max_count**
  - **topic_data_max_length**
  - **topic_data_max_count**
  - **publisher_group_data_max_length**
  - **publisher_group_data_max_count**
  - **subscriber_group_data_max_length**

- **subscriber_group_data_max_count**
- **writer_user_data_max_length**
- **writer_user_data_max_count**
- **reader_user_data_max_length**
- **reader_user_data_max_count**

See the *User Discovery Data* chapter in the User's Manual for more information on these QoS policies.

#### Support for environment variable expansion in MAG

Now you can refer to an environment variable set in the command shell within an XML tag. When MAG parses the configuration file, it will expand the environment variable. The way to refer to the environment variable is as follows:

```
$(MY_VARIABLE)
```

For example:

```
<name>$(MY_VARIABLE)</name>
```

Being able to refer to an environment variable within an XML file increases XML reusability. For example, this will allow you to specify the initial peers, so you do not need to use multiple XML files or XML profiles per application.

#### Only check for QoS policies that are used by your system definition

In previous releases, MAG checked whether all of the QoS policies passed to the tool were supported by *Connext Micro*. This has been changed to only check for QoS policies that are used by the system defined in the `<domain_participant_library>`.

#### XML fields of type duration have unset tags default to 0 with a warning log message

The duration type tag has two subfields, `<sec>` and `<nanosec>`. Some QoS policies that use these fields, such as the DEADLINE QoS Policy, set the default duration to INFINITE. Therefore, if you had set just one of these fields (such as `<sec>`, but not `<nanosec>`, or vice-versa), the resulting duration value was still INFINITE.

Now if you set only one of these fields (`<sec>` or `<nanosec>`) in the XML file, the other value defaults to 0. (If you set neither one of them, the default duration for that policy would be used.) A warning message will also be logged by the parser specifying the parent tag, the missing subfield, and the line number.

#### Support for resource limits in DomainParticipantFactoryQos

This release allows you to configure the resource limits of the DomainParticipantFactoryQos (**max_participants**) in XML.

By default, MAG updates the resource limits of the DomainParticipantFactoryQos so that MAG can at least support the entities defined in the XML file. However, if your applications communicate with more remote entities than those specified in the XML file, you may need to manually update the resource limits. In that case, you need to use the `-dontUpdateResourceLimits` command-line option. That will prevent MAG from automatically updating the resource limits for the *DomainParticipantFactory*, *DomainParticipants*, *DataReaders*, and *DataWriters*.

#### Instance replacement changes affect XML files in MAG

The type used by `<instance_replacement>` in MAG has been changed from a single type to a complex type. Because of this change, XML files used by MAG in previous releases won't work out of the box in this release. For example, the following XML based on MAG in previous releases won't work in the current release:

```
<datareader_qos>
    <reader_resource_limits>
        <instance_replacement>OLDEST_INSTANCE_REPLACEMENT</instance_replacement>
    </reader_resource_limits>
</datareader_qos>
```

You need to update it to the following:

```
<datareader_qos>
    <reader_resource_limits>
        <instance_replacement>
            <alive_instance_removal>ANY_INSTANCE_REMOVAL</alive_instance_removal>
            <disposed_instance_removal>ANY_INSTANCE_REMOVAL</disposed_instance_removal>
            <no_writers_instance_removal>ANY_INSTANCE_REMOVAL</no_writers_instance_
↪removal>
        </instance_replacement>
    </reader_resource_limits>
</datareader_qos>
```

### 10.5.8 What's Fixed in 4.0.0

The following are fixes since *Connext Micro* 3.0.3.

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

### Discovery

#### [Critical] Failure to interoperate with other DDS implementations if default multicast locator specified

*Connext Micro* did not interoperate with other DDS implementations when the default multicast locator was specified.

[RTI Issue ID MICRO-5148]

#### [Major] Incorrect lease_duration may have been used for a discovered participant.

In previous releases, if the **lease_duration** was not sent by a remote *DomainParticipant*, a previously received value was used instead.

Note that RTI's DDS implementations send the **lease_duration**.

[RTI Issue ID MICRO-3254]

### Serialization and Deserialization

#### [Critical] DataReader on a Topic using an appendable type may receive samples with incorrect value

A *DataReader* subscribing to a *Topic* on an appendable type may have received incorrect samples from a matching *DataWriter*.

The problem only occurred when the *DataWriter* published a type with fewer members than the *DataReader* type. For example, consider a *DataWriter* on FooBase and a *DataReader* on FooDerived:

```
@appendable struct FooBase {
    sequence<uint8,1024>base_value;
};

@appendable struct FooDerived {
    sequence<uint8,1024> base_value;
    @default(12) uint8 derived_value;
};
```

In this case, the serialized sample stream would be padded with extra bytes to align the stream to 4 bytes as required by the OMG Extensible and Dynamic Topic Types for DDS specification, version 1.3. However, the additional padding bytes were incorrectly interpreted as part of the data and `derived_value` may have been set to a random value.

For example, in the case above, when the *DataWriter* published a sample with type FooBase, in some cases the *DataReader* received a sample in which the field `derived_value` was set to 0 instead of 12.

**Note:** *Connext Micro* does not support the `@default` annotation.

[RTI Issue ID MICRO-6402]

#### [Critical] Malformed samples with invalid strings not dropped by DataReader

A *DataReader* may have provided the application a malformed sample containing an invalid value (not Null-terminated) for a string member. The string member may not have been Null-terminated, resulting in undefined behavior if the application tried to access it.

Now, the *DataReader* will not deserialize the sample and the sample will not be provided to the application.

[RTI Issue ID MICRO-3039]

#### [Major] Float and double ranges may not have been enforced correctly

Float and double ranges may not have been enforced correctly. Float and double member values that should not have passed the check ended up passing it.

This issue only occurred under any of the following conditions:

For "float":

- When @min was set to -3.4E38 for a member, a value smaller than @min passed the check when it should not have.
- When @max was set to 3.4E38 for a member, a value greater than @max passed the check when it should not have.

For "double":

- When @min was set to -1.7E+308 for a member, a value smaller than @min passed the check when it should not have.
- When @max was set to 1.7E+308 for a member, a value greater than @max passed the check when it should not have.

For "float" and "double":

- When the member value was set to INFINITY, samples passed the range check when they should not have.
- When the member value was set to NaN, samples passed the range check when they should not have.

[RTI Issue ID MICRO-3280]

#### [Major] Deserialization of tampered/corrupted samples may have unexpectedly succeeded

A *DataReader* may not have detected that a truncated sample due to corruption or tampering was invalid. As a result, the application may have received samples with invalid content.

Now, the deserialization of corrupted samples fails, and they are not provided to the application.

[RTI Issue ID MICRO-3057]

#### [Major] Invalid serialization of samples with types containing nested structures with primitive members that require padding

In *Connext DDS* 6.0.1 and earlier, the serialization of samples with a type containing two or more levels of nested complex types, where the nested types have primitive members that require padding, may have failed. This means that a *DataReader* may have received an invalid value for a sample. Example:

```
// Level-2 Nested type
   struct Struct1 {
      uint8 m1;
      uint8 m2;
      int32 m3;
   };

   // Level-1 Nested type
   struct Struct2 {
      int32 m1;
      int32 m2;
      uint8 m3;
      uint8 m4;
      Struct1 m5;
   };

   struct Struct3 {
      Struct2 m1;
   };
```

In the above example, Struct2 and Struct1 are nested, and there is padding between Struct1::m2 (1-byte aligned) and Struct1::m3 (4-byte aligned) of 2 bytes.

This issue only applied to nested types that are appendable or final for XCDR1 data representation or final for XCDR2 data representation.

This problem affected DynamicData and the generated code for the following languages: C, C++, C++03, and C++11.

For generated code, a potential workaround to this problem was to generate code with a value of 1 or 0 for the -optimization, but this may have had performance implications.

[RTI Issue ID MICRO-2744]

#### [Minor] Serialization of string members did not check for null-terminated strings in C, traditional C++, and modern C++

The code executed by a *DataWriter* that serializes string members in a Topic type did not check that the strings are null-terminated. This may have led to undefined behavior, because the serialization code calls strlen.

This problem has been fixed. The serialization code now checks for null-terminated strings with the maximum allowed length and reports the following error if the string is not well-formed:

```
RTIXCdrInterpreter_serializeString:StrStruct:member2 serialization error. String length␣
↪(at least 6) is larger than maximum 5
```

[RTI Issue ID MICRO-3040]

### Usability

#### [Trivial] Thread names were not set on QNX

In previous releases, the thread names were not set on QNX.

[RTI Issue ID MICRO-5851]

### Transports

#### [Critical] Stalled communication when using shared-memory transport

On systems with a weak memory architecture, such as Arm®, the shared-memory (SHMEM) transport may have been corrupted due to a data race in the concurrent queue where the messages are written into the shared-memory segment. This data race may have occurred until **received_message_count_max messages** were sent through the transport. The corrupted transport resulted in parsing errors, which filled up the shared-memory segment, stalling communication.

[RTI Issue ID MICRO-5931]

#### [Critical] Undefined behavior when using SHMEM transport in Linux, macOS, QNX, Integrity, and Lynx

There was an issue in the shared-memory (SHMEM) transport implementation that may have led to undefined behavior in your *Connext Micro* application, including data corruption, errors, and hangs. The problem could occur in Linux®, macOS®, QNX®, INTEGRITY®, and LynxOS® systems.

[RTI Issue ID MICRO-5932]

### Reliability Protocol and Wire Representation

#### [Critical] Reliable DataWriter may have ignored requests to resend samples

If a *DataWriter* received multiple requests to resend samples before its periodic heartbeat period expired, the *DataWriter* may have ignored the request if the requested sample had been sent and was also the first expected sample by the requesting *DataReader*.

[RTI Issue ID MICRO-5183]

#### [Minor] Incorrect heartbeat sent before first sample when first_write_sequence_number is different from 1

In previous releases, if the **DataWriterQos.protocol.rtps_reliable_writer.first_write_sequence_number** was different from the default value 1, heartbeats sent before the first sample was written would indicate 1 as the first sample available. This caused a *DataReader* to wait for samples with a sequence number less than **DataWriterQos.protocol.rtps_reliable_writer.first_write_sequence_number** until a heartbeat with the correct first sequence number was received.

[RTI Issue ID MICRO-4081]

### Logging

#### [Major] Race condition and memory corruption in logger

The following issues have been fixed in the logger:

- Processing log-messages in a log handler was not thread-safe.
- Memory corruption may have occurred.
- Conversion of INT_MIN was incorrect.

---

**Note:** The `OSAPI_Log_clear` API must not be called outside a log-handler since it is no longer thread-safe.

---

[RTI Issue ID MICRO-5854]

### Performance and Scalability

#### [Trivial] Asynchronous publication delay

In previous releases, there was a delay (equal to the OSAPI Task scheduler's clock rate) before sending a fragmented or flow-controlled sample. This delay has been removed.

[RTI Issue ID MICRO-5853]

### APIs (C or Traditional C++)

#### [Critical] Segmentation fault when finalizing DataWriter QoS

Finalizing a *DataWriter* QoS could have resulted in a segmentation fault if **publish_mode.name** was set to a builtin Flow Controller name.

[RTI Issue ID MICRO-5966]

#### [Major] DDS_Subscriber_lookup_datareader may return a DataReader that was created by a different Subscriber

The `DDS_Subscriber_lookup_datareader` API searches for a *DataReader* for a given TopicDescription created by the *Subscriber*. However, in previous releases, it the returned *DataReader* could belong to a different *Subscriber* if multiple *DataReaders* were created for the same *Topic* in different *Subscribers*.

[RTI Issue ID MICRO-4569]

#### [Major] DDS_Publisher_lookup_datawriter may return a DataWriter that was created by a different Publisher

The `DDS_Publisher_lookup_datawriter` API searches for a *DataWriter* for a given *Topic* created by the *Publisher*. However, in previous releases, the returned *DataWriter* could belong to a different *Publisher* if multiple *DataWriters* were created for the same *Topic* in different *Publishers*.

[RTI Issue ID MICRO-4570]

#### [Major] DDS_Entity_enable was not thread-safe for a DomainParticipant

*DDS_Entity_enable* was not thread-safe, which may have led to race conditions.

[RTI Issue ID MICRO-3379]

#### [Major] Race condition in DDS enable APIs

A race condition existed if the same DDS entity was enabled from multiple threads at the same time.

[RTI Issue ID MICRO-3311]

#### [Minor] DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT ignored when used as argument to DDS_DomainParticipant_create_flowcontroller

The following related issues are resolved in this release:

- *Connext Micro* ignored DDS_FLOW_CONTROLLER_PROPERTY_DEFAULT when passed in to the `DDS_DomainParticipant_create_flowcontroller` API call.
- The properties used by *Connext Micro* for the builtin Flow Controllers were not aligned with *Connext Professional*.
- The default Flow Controller properties returned were not aligned with *Connext Professional*.

[RTI Issue ID MICRO-6118]

#### [Minor] Failure to parse invalid index

A peer descriptor string consisting of only an invalid range, e.g, "`[3`" was incorrectly interpreted as the empty peer address string "".

[RTI Issue ID MICRO-4436]

### Generated Code (C, Traditional C++, and Modern C++)

#### [Critical] Foo_create_data() failed to create samples for data types with long doubles

`Foo_create_data()` failed to create samples of types that contained arrays or sequences of types that contained long doubles. For example, `Foo_create_data()` failed for the following type Foo:

```
struct S
{
    long double ld;
};

struct Foo
{
    sequence<S> s;
};
```

However, `Foo_create_data()` did not fail for the following type Foo:

```
struct Foo
{
    sequence<long double> s;
};
```

[RTI Issue ID MICRO-3025]

#### [Minor] Example code generated from XML or XSD files failed to compile

Example code generated by *rtiddsgen* from XML or XSD files failed to compile.

[RTI Issue ID MICRO-2505]

### Micro Application Generator

#### [Major] NullPointerException when using -outputFinalQoS if QoS Profile did not define each internal QoS

When using MAG with the `-outputFinalQoS` option, if the QoS Profile to check did not contain a definition of each internal QoS (participant_qos, publisher_qos, etc.) directly or by inheriting from another QoS Profile, MAG reported this error:

```
Exception in thread "main" java.lang.NullPointerException
   at com.rti.micro.appgen.utils.QosUtils.removeNullElementsFromList(QosUtils.java:2332)
   at com.rti.micro.appgen.utils.QosUtils.removeNullElements(QosUtils.java:2256)
   at com.rti.micro.appgen.MicroAppGen.main(MicroAppGen.java:328)
```

[RTI Issue ID MAG-121]

#### [Minor] MAG failed to generate code when qos_profile inherited from individual QoS policies

MAG failed to generate code when a `<qos_profile>` inherited from individual QoS policies. For example, running MAG with the following input file caused an error:

```
<qos_library name="QosLibrary">
   <qos_profile name="QosProfile1" is_default_qos="true">
      <participant_qos name="QosParticipant">
         ...
      </participant_qos>
   </qos_profile>
   <qos_profile name="QosProfile2" base_name="QosProfile1::QosParticipant">
   </qos_profile>
</qos_library>
```

The error was:

```
...
11:31:40.548 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to calculate the␣
↪system model.
java.lang.Exception: Unable to find QoS library/profile 'QosProfile1::QosParticipant'.
...
11:31:40.552 [main] INFO com.rti.micro.appgen.MicroAppGen - Exiting.
```

Now MAG properly handles this case.

[RTI Issue ID MAG-105]

#### [Minor] MAG always used default value for disable_auto_interface_config

MAG always used the default value for `disable_auto_interface_config` in the generated code, regardless of the value specified in the XML.

[RTI Issue ID MAG-110]

#### [Minor] MAG failed if arguments contained whitespace on Linux systems

On Linux systems, MAG failed to run if any arguments contained whitespace. It logged an error similar to the following:

```
12:04:55.205 [main] ERROR com.rti.micro.appgen.MicroAppGen - Only 1 input file
can be processed.
12:04:55.208 [main] INFO  com.rti.micro.appgen.MicroAppGen - Exiting.
```

[RTI Issue ID MAG-118]

#### [Trivial] XSD validation failed if flags used a combination of values

The XSD validation of an XML application file failed if there was a UDPv4 configuration using a combination of values for the `flags` element. For example, this snippet caused an error:

```
<transport_builtin>
   <udpv4>
         <interface_table>
            <element>
               <flags>
                     UDP_INTERFACE_INTERFACE_UP_FLAG|UDP_INTERFACE_INTERFACE_MULTICAST_
↪FLAG
               </flags>
            </element>
         </interface_table>
   </udpv4>
</transport_builtin>
```

The error was:

```
ERROR com.rti.micro.appgen.MicroAppGen - cvc-pattern-valid:
Value 'UDP_INTERFACE_INTERFACE_UP_FLAG|UDP_INTERFACE_INTERFACE_MULTICAST_FLAG'
is not facet-valid with respect to pattern
'(UDP_INTERFACE_INTERFACE_UP_FLAG|UDP_INTERFACE_INTERFACE_MULTICAST_FLAG)'
for type 'udpInterfaceFlagMask'.
```

Now combinations are allowed.

[RTI Issue ID MAG-114]

### OMG Specification Compliance

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

#### [Major] DDS_StatusCondition_set_enabled_statuses did not trigger if an active condition was enabled and had incorrect default value

In previous releases, if a StatusCondition enabled by a call to `DDS_StatusCondition_set_enabled_statuses` was already active, the StatusCondition did not trigger.

The default enabled status list was incorrectly set to DDS_STATUS_MASK_NONE, but is now set to DDS_STATUS_MASK_ALL until the first successful call to `DDS_StatusCondition_set_enabled_statuses`.

[RTI Issue ID MICRO-3308]

### Interoperability

#### [Critical] Failure to deserialize fragmented samples sent by Connext Professional 7

Due to incorrect processing of an inline QoS in a fragmented sample, *Connext Micro* failed to deserialize fragmented samples sent by *Connext Professional 7*, or other implementations that set the length of the PID_SENTINEL to a value different than 1.

[RTI Issue ID MICRO-4095]

#### [Critical] Inline QoS offset non-compliant with DDSI-RTPS standard

The inline QoS offset in DATA was set to 0 instead of the offset to the next field after the inline QoS. This may have caused DDS implementations from other vendors to fail to receive data.

[RTI Issue ID MICRO-4160]

#### [Critical] Connext Micro may have repeated requesting a sample that was no longer available from a DataWriter

If *Connext Micro* detects a missing sample when using DDS_RELIABLE_RELIABILITY_QOS reliability, it will request the sample to be resent, but if the sample is no longer available from the *DataWriter*, the *DataWriter* may send a GAP message to indicate the sample is not longer available.

*Connext Micro* failed to interpret the GAP message correctly if the first sequence number in the GAP message was equal to the bitmap base of the GAP message. In this case, *Connext Micro* failed to ignore the no-longer-available sample and kept sending a request for the sample.

[RTI Issue ID MICRO-4668]

#### [Critical] Failure to deserialize a fragmented sample with multiple fragments in a DATA_FRAG submessage

A deserialization error occurred when deserializing a sample that was fragmented into multiple fragments in a single RTPS DATA_FRAG submessage.

[RTI Issue ID MICRO-2958]

### Vulnerabilities

#### [Critical] Vulnerabilities in RTI Micro Application Generator (MAG)

This release fixes vulnerabilities in Log4j known as "log4shell". You can find further details in RTI's **Security Notice 2021-12-log4j** at https://community.rti.com/kb/apache-log4j-vulnerability-cve-2021-44228cve-2021-45046-impact-rti-connext-products.

RTI Micro Application Generator uses Apache Log4j version 2.17.1 in this release.

[RTI Issue ID MAG-147]

#### [Critical] Illegal memory access when failing to generate interpreter programs

Receiving malicious endpoint discovery information might have resulted (very rarely) in an arbitrary read from the thread stack.

User impact with or without security was as follows:

- Remotely exploitable
- Crash application
- Potentially impacting confidentiality of Connext application
- CVSS Base Score: 6.5 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:H

[RTI Issue ID MICRO-3219]

#### [Critical] Potential crash when receiving a malformed sample using DDS_XCDR2_DATA_REPRESENTATION

A *Connext Micro* application could have crashed if a *DataReader* received a malformed serialized sample using DDS_XCDR2_DATA_REPRESENTATION. The issue only affected appendable or mutable types.

User impact with or without security was as follows:

- Remotely exploitable through malicious RTPS messages
- Connext application could crash or potentially leak sensitive information
- CVSS Base Score: 6.5 MEDIUM
- CVSS v3.1 Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:H

[RTI Issue ID MICRO-3118]

### Other

#### [Minor] Delay in sending data when using a flow-controller

When using a flow-controller to send data, there was a delay before sending the first sample or fragment (of up to one task period).

[RTI Issue ID MICRO-6494]

#### [Minor] Non-default timer resolutions may have caused an incorrect timeout

Compiling *Connext Micro* with a non-default timer resolution may have caused incorrect timeouts.

[RTI Issue ID MICRO-6476]

## 10.6 Known Issues

**Note:** For an updated list of critical known issues, see the Critical Issues List on the RTI Customer Portal at https://support.rti.com.

### 10.6.1 Possible memory leak when entity creation functions fail

If DDS entity creation fails due to a configuration error, a memory leak may occur.

[RTI Issue ID MICRO-3531]

### 10.6.2 DomainParticipant_assert_liveliness does not function properly if Reader/Writer are in same DomainParticipant

`DomainParticipant_assert_liveliness` does not properly assert liveliness if the *DataReader* and *DataWriter* are in the same *DomainParticipant*.

[RTI Issue ID MICRO-2759]

### 10.6.3 GCC warnings caused by type plugin functions

When compiling `TypePluginProgramExecution.c` (only included in the full source bundle), GCC version 12 and above emits the following warning:

```
storing the address of local variable 'xcdrStream' in '*localContext.xcdrStream' [-
↪Wdangling-pointer=]

    localContext->xcdrStream = xcdrStreamPtr;
```

This warning is benign and the behavior is correct.

[RTI Issue ID MICRO-13220]

### 10.6.4 MAG always generates initial value for AUTOSAR data type with one element

Due to Micro Application Generator's limited capacity to process data types, the initial value generated for `AUTOSAR_CDD` only contains a single element, regardless of the number of elements required as defined in DDS-XML. This can cause incompatibility errors in DaVinci Developer.

As a workaround, modify the generated files `application.arxml` and `DdsCddType.arxml` to add values as needed, either manually or through DaVinci Developer.

[RTI Issue ID MICRO-13486]

### 10.6.5 MAG does not support hex values as XML input

Micro Application Generator (MAG) cannot process hex inputs from XML files.

As a workaround, convert hex values to decimal values in your XML file before running *rtiddsmag*.

[RTI Issue ID MAG-402]

### 10.6.6 Samples cannot be recovered if subscribing application fails to return loan

When a subscribing application has taken a loan for the Zero Copy v2 transport (using the API FooDataReader_read() or FooDataReader_take()) and fails to return the loan due to a crash or other circumstances, *Connext Micro* cannot recover those samples. This also affects the matching *DataWriter*, which cannot reclaim the samples and continues to run in a degraded state.

[RTI Issue ID MICRO-5834]

### 10.6.7 64-bit discriminator values greater than (2^31-1) or smaller than (-2^31) not supported

Unions with a 64-bit integer discriminator type containing discriminator values that cannot fit in a 32-bit value are not supported when using the following language bindings:

- C
- Traditional C++

They are also not supported with ContentFilteredTopics, regardless of the language binding.

Using label values greater than 32-bit may lead to receiving samples with invalid content or to filtering samples incorrectly.

[RTI Issue ID MICRO-3056]

### 10.6.8 NaN and INF float and doubles are not detected and will not cause errors

Normally, *Connext Micro* discards samples with values that are out of range during serialization and de-serialization; however, Not a Number (NaN) and Infinite (INF) floating point and doubles are not detected and will not cause serialization or de-serialization errors.

[RTI Issue ID MICRO-5960]

### 10.6.9 Ungracefully terminated QNX processes using SHMEM transport prevents startup of new processes due to unclosed POSIX semaphores

If a QNX 7.0 or earlier application using the shared-memory transport was ungracefully shut down, crashed, or otherwise had an abnormal termination while holding a POSIX semaphore used by the transport (for example, while sending data through the shared-memory transport), *Connext* applications launched after that point on the same domain may wait forever for that semaphore to be released.

Workaround for QNX 7.0 and earlier: to enable new applications to start, RTI recommends stopping all applications, then cleaning up the Inter-Process Communication (IPC) resources before starting new applications.

This problem is resolved for QNX 7.1, as described in the fix for *[Critical] Ungracefully terminated QNX processes using SHMEM transport prevented startup of new processes due to unclosed POSIX semaphores*.

[RTI Issue ID MICRO-6013]

### 10.6.10 Flow Controllers require RTOS

Flow controllers require an RTOS.

[RTI Issue ID MICRO-6648]

### 10.6.11 Porting Guide not included

Starting in release 4.2.0, *Connext Micro* has many internal changes from previous versions of *Connext Micro*, and APIs are not considered stable. Therefore, instructions for porting *Connext Micro* are not included. If you need instructions to port *Connext Micro*, please contact support@rti.com.

[RTI Issue ID MICRO-8618]

### 10.6.12 Platform Independent Library toolchain dependencies

The platform independent libraries (PIL) are not completely independent of the toolchain and standard C library, and thus require a compatible toolchain and standard library to link to. See *Platforms Guide* for more information.

[RTI Issue ID MICRO-8154]

### 10.6.13 Technical limitations for semaphore timeout variance on Windows systems

The Windows semaphore implementation utilizes the kernel-level `WaitForSingleObject` function. While the Windows API represents time in 100ns intervals (ticks), the Windows kernel dispatcher does not evaluate wait conditions at that resolution. Instead, thread completion and timeout expiration are governed by the system interrupt timer.

In standard Windows configurations for *Connext Micro*, this interrupt occurs every 15.6ms. Because the kernel processes the "waitlist" only during these interrupt intervals, a requested timeout is rounded to the nearest interrupt boundary. This creates a scenario where a thread may be signaled at the start of the interrupt period encompassing the target time, resulting in a return that is physically "early" when measured against a high-resolution clock.

[RTI Issue ID MICRO-13073]

## 10.7 Experimental Features

This software may contain experimental features. These are used to evaluate potential new features and obtain customer feedback. They are not guaranteed to be consistent or supported and they should not be used in production.

In the API Reference HTML documentation, experimental APIs are marked with **«experimental»**.

Experimental features are also clearly noted as such in the *User's Manual* or *Getting Started Guide* for the component in which they are included.

Disclaimers:

- Experimental features may be only available in a subset of the supported languages and for a subset of the supported platforms.
- Experimental features may change in the future.
- Experimental features may or may not appear in future product releases.
- Experimental features should not be used in production.

Please submit your comments and suggestions about experimental features to **support@rti.com** or via the RTI Customer Portal (https://support.rti.com/).

# Chapter 11

# Benchmarks

Performance benchmarks are no longer included with an *RTI Connext Micro* installation. Please refer to the RTI Connext Performance Benchmarks on RTI Community for more information.

---

**Note:** The *RTI Connext Performance Benchmarks* contain metrics for multiple products and versions, so please ensure that you refer to the appropriate section.

---

# Chapter 12

# Copyrights

© 2017-2026 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
June 2026.

### Trademarks

RTI, Real-Time Innovations, Connext, Connext Drive, NDDS, the RTI logo, 1RTI and the phrase, "Your Systems. Working as one." are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

### Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished solely under and subject to RTI's standard terms and conditions available at https://www.rti.com/terms and in accordance with your License Acknowledgement Certificate (LAC) and Maintenance and Support Certificate (MSC), except to the extent otherwise agreed to in writing by RTI.

### Third-Party Software

RTI software may contain independent, third-party software or code that are subject to third-party license terms and conditions, including open source license terms and conditions. Copies of applicable third-party licenses and notices are located at community.rti.com/documentation. IT IS YOUR RESPONSIBILITY TO ENSURE THAT YOUR USE OF THIRD-PARTY SOFTWARE COMPLIES WITH THE CORRESPONDING THIRD-PARTY LICENSE TERMS AND CONDITIONS.

## Notices

*Deprecations and Removals*

Any deprecations or removals noted in this document serve as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software.

*Deprecated* means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in a release, RTI hereby provides customer notice that RTI reserves the right after one year from the date of such release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

Technical Support
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

# Chapter 13

# Third-Party and Open Source Software

This section outlines Real-Time Innovations (RTI) usage of first-level third-party and open source software in the *RTI Connext Micro* libraries and utilities.

## 13.1 Connext Micro Libraries

### 13.1.1 fnmatch

- Related to: Content-filtered topics, query conditions, partitions, multicast address management, topic filter in XML QoS profile.
- Software is included in the core middleware libraries.
- Third-Party Software License:

```
Copyright (c) 1989, 1993, 1994

The Regents of the University of California. All rights reserved.

This code is derived from software contributed to Berkeley by Guido
van Rossum.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
```

(continues on next page)

(continued from previous page)

```
3. All advertising materials mentioning features or use of this
software must display the following acknowledgement:

This product includes software developed by the University of
California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS \``AS IS''
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

### 13.1.2 crc32c.c

- Related to: RTPS CRC-32 checksum support
- Version 1.1
- Third-Party Software License:

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the author be held liable for any damages
arising from the use of the software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
claim that you wrote the original software. If you use this software
in a product, an acknowledgement in the product documentation would
be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not
be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
distribution
```

(continues on next page)

(continued from previous page)

```
Mark Adler

madler@alumni.caltech.edu
```

### 13.1.3 MD5

- Related to: DDS keys implementation, content-filtered topics (to sign the filter), persistence service (to generate writer-side unique identification), Integration Toolkit for AUTOSAR (DDS-IDL Service Interface code generation)
- Software is included in core DDS middleware libraries, in the Connext DDS Micro libraries, in the Connext DDS Cert libraries and in the Integration Toolkit for AUTOSAR.
- Third-Party Software License:

```
Copyright (C) 1999, 2002 Aladdin Enterprises.  All rights reserved.

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
    claim that you wrote the original software. If you use this software
    in a product, an acknowledgment in the product documentation would be
    appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
    misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

L. Peter Deutsch
ghost@aladdin.com
```

## 13.2 RTI Code Generator (rtiddsgen)

### 13.2.1 ANTLR

- This software is distributed with rtiddsgen (RTI Code Generator) as a jar file. The source code is not modified or shipped. In addition, the output produced by this software from a grammar file is part of the rtiddsgen JAR file.
- Version: Release 3.5.2
- Open Source Software License: ANTLR 3 License (full text found in the *Licenses*).

### 13.2.2 ANTLR Runtime API

- Used in rtiddsgen.
- Version: Release 3.5.2
- Open Source Software License: ANTLR 3 License (full text found in the *Licenses*).

### 13.2.3 Apache Commons CLI

- Used in rtiddsgen.
- Version: 1.5.0
- Open Source Software License: Apache License Version 2.0 (full text found in the *Licenses*).

### 13.2.4 Apache Commons Lang

- Used in rtiddsgen. We only use the class StringUtils.
- Version 2.6
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.2.5 Apache Commons Lang 3

- Used in rtiddsgen.
- Version 3.11
- Open Source Software License: Apache License Version 2.0 (full text found in the *Licenses*).

### 13.2.6 Apache Log4j 2

- This software is distributed with rtiddsgen (RTI Code Generator) as a jar file. The source code is not modified or shipped.
- Source: https://logging.apache.org/
- Version: 2.25.3
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.2.7 Apache Velocity

- This software is included in rtiddsgen. The source code is not modified or shipped.
- Source: http://velocity.apache.org/
- Version: 2.3
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.2.8 Simple Logging Facade for Java (SLF4J)

- This software is included in rtiddsgen.
- Version 1.7.35
- Open Source Software License: https://www.slf4j.org/license.html

```
Copyright (c) 2004-2025 QOS.ch
All rights reserved.

Permission is hereby granted, free  of charge, to any person obtaining
a  copy  of this  software  and  associated  documentation files  (the
"Software"), to  deal in  the Software without  restriction, including
without limitation  the rights to  use, copy, modify,  merge, publish,
distribute,  sublicense, and/or sell  copies of  the Software,  and to
permit persons to whom the Software  is furnished to do so, subject to
the following conditions:

The  above  copyright  notice  and  this permission  notice  shall  be
included in all copies or substantial portions of the Software.

THE  SOFTWARE IS  PROVIDED  "AS  IS", WITHOUT  WARRANTY  OF ANY  KIND,
EXPRESS OR  IMPLIED, INCLUDING  BUT NOT LIMITED  TO THE  WARRANTIES OF
MERCHANTABILITY,    FITNESS    FOR    A    PARTICULAR    PURPOSE    AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE,  ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

### 13.2.9 Gson

- Portions of rtiddsgen are built using Gson.
- Version 2.9.1
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

## 13.3 Micro Application Generator (rtiddsmag)

### 13.3.1 Apache Commons CLI

- Used in Micro Application Generator.
- Version 1.4
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.3.2 Apache Commons Lang 3

- Used in Micro Application Generator. We only use the class StringUtils.
- Version 3.7
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.3.3 Apache Log4j 2

- This software is distributed with rtiddsmag (Micro Application Generator) as a jar file. The source code is not modified or shipped.
- Source: https://logging.apache.org/
- Version: 2.25.3
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.3.4 Apache Velocity

- This software is included in rtiddsmag (Micro Application Generator). The source code is not modified or shipped.
- Source: http://velocity.apache.org/
- Version: 2.0
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

### 13.3.5 Istack Common Utility Code Runtime

- Version 3.0.7
- Open Source Software License: Common Development and Distribution License (CDDL) Version 1.1 (full text found in *Licenses*).

### 13.3.6 JavaBeans Activation Framework API

- Version 1.2.0
- Open Source Software License: Common Development and Distribution License (CDDL) Version 1.1 (full text found in *Licenses*).

### 13.3.7 Javax Annotation API

- Version 1.3.2
- Open Source Software License: Common Development and Distribution License (CDDL) Version 1.1 (full text found in *Licenses*).

### 13.3.8 JAXB API

- Version 2.3.1
- Open Source Software License: Common Development and Distribution License (CDDL) Version 1.1 (full text found in *Licenses*).

### 13.3.9 JAXB Runtime

- Version 2.3.9
- Open Source Software License: Common Development and Distribution License (CDDL) Version 1.1 (full text found in *Licenses*).

### 13.3.10 Simple Logging Facade for Java (SLF4J)

- This software is included in rtiddsmag (Micro Application Generator).
- Version 1.7.25
- Open Source Software License: https://www.slf4j.org/license.html

```
Copyright (c) 2004-2025 QOS.ch
All rights reserved.

Permission is hereby granted, free  of charge, to any person obtaining
a  copy  of this  software  and  associated  documentation files  (the
"Software"), to  deal in  the Software without  restriction, including
without limitation  the rights to  use, copy, modify,  merge, publish,
distribute,  sublicense, and/or sell  copies of  the Software,  and to
permit persons to whom the Software  is furnished to do so, subject to
the following conditions:

The  above  copyright  notice  and  this permission  notice  shall  be
included in all copies or substantial portions of the Software.
```

(continues on next page)

(continued from previous page)

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

### 13.3.11 TXW2

- Version 2.3.1
- Open Source Software License: Common Development and Distribution License (CDDL) Version 1.1 (full text found in *Licenses*).

## 13.4 Lightweight Security Plugin

### 13.4.1 OpenSSL

- Related to: cryptographic functionality
- Software is provided as separate libraries. RTI Security-related plugins use public OpenSSL APIs. We did not copy-paste OpenSSL internal code. RTI Security-related plugins depend on the OpenSSL libraries.
- Release: 3.5.5
- Download link: https://github.com/openssl/openssl/releases/download/openssl-3.5.5/openssl-3.5.5.tar.gz
- Open Source Software License: Apache License Version 2.0 (full text found in *Licenses*).

## 13.5 Licenses

### 13.5.1 ANTLR 3 License

```
[The BSD License]

Copyright (c) 2010 Terence Parr

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
```

(continues on next page)

(continued from previous page)

```
are met:

Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

Neither the name of the author nor the names of its contributors may
be used to endorse or promote products derived from this software
without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

### 13.5.2 Apache License version 2.0, January 2004 (http://www.apache.org/licenses/)

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
```

(continues on next page)

(continued from previous page)

```
    exercising permissions granted by this License.

    "Source" form shall mean the preferred form for making modifications,
    including but not limited to software source code, documentation
    source, and configuration files.

    "Object" form shall mean any form resulting from mechanical
    transformation or translation of a Source form, including but
    not limited to compiled object code, generated documentation,
    and conversions to other media types.

    "Work" shall mean the work of authorship, whether in Source or
    Object form, made available under the License, as indicated by a
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    "Derivative Works" shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    "Contribution" shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, "submitted"
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
```

(continues on next page)

(continued from previous page)

```
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
```

(continues on next page)

(continued from previous page)

```
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

(continues on next page)

(continued from previous page)

```
   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```

### 13.5.3 Common Development and Distribution License (CDDL) Version 1.1

```
1. Definitions.

  1.1. "Contributor" means each individual or entity that creates or
  contributes to the creation of Modifications.

  1.2. "Contributor Version" means the combination of the Original
  Software, prior Modifications used by a Contributor (if any), and
  the Modifications made by that particular Contributor.

  1.3. "Covered Software" means (a) the Original Software, or (b)
  Modifications, or (c) the combination of files containing Original
  Software with files containing Modifications, in each case including
  portions thereof.

  1.4. "Executable" means the Covered Software in any form other than
  Source Code.

  1.5. "Initial Developer" means the individual or entity that first
  makes Original Software available under this License.

  1.6. "Larger Work" means a work which combines Covered Software or
  portions thereof with code not governed by the terms of this License.
```

(continues on next page)

(continued from previous page)

```
  1.7. "License" means this document.

  1.8. "Licensable" means having the right to grant, to the maximum
  extent possible, whether at the time of the initial grant or
  subsequently acquired, any and all of the rights conveyed herein.

  1.9. "Modifications" means the Source Code and Executable form of
  any of the following:

  A. Any file that results from an addition to, deletion from or
  modification of the contents of a file containing Original Software
  or previous Modifications;

  B. Any new file that contains any part of the Original Software or
  previous Modification; or

  C. Any new file that is contributed or otherwise made available
  under the terms of this License.

  1.10. "Original Software" means the Source Code and Executable form
  of computer software code that is originally released under this
  License.

  1.11. "Patent Claims" means any patent claim(s), now owned or
  hereafter acquired, including without limitation, method, process,
  and apparatus claims, in any patent Licensable by grantor.

  1.12. "Source Code" means (a) the common form of computer software
  code in which modifications are made and (b) associated
  documentation included in or with such code.

  1.13. "You" (or "Your") means an individual or a legal entity
  exercising rights under, and complying with all of the terms of,
  this License. For legal entities, "You" includes any entity which
  controls, is controlled by, or is under common control with You. For
  purposes of this definition, "control" means (a) the power, direct
  or indirect, to cause the direction or management of such entity,
  whether by contract or otherwise, or (b) ownership of more than
  fifty percent (50%) of the outstanding shares or beneficial
  ownership of such entity.

2. License Grants.

  2.1. The Initial Developer Grant.

  Conditioned upon Your compliance with Section 3.1 below and subject
  to third party intellectual property claims, the Initial Developer
  hereby grants You a world-wide, royalty-free, non-exclusive license:

  (a) under intellectual property rights (other than patent or
```

(continues on next page)

(continued from previous page)

```
trademark) Licensable by Initial Developer, to use, reproduce,
modify, display, perform, sublicense and distribute the Original
Software (or portions thereof), with or without Modifications,
and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of
Original Software, to make, have made, use, practice, sell, and
offer for sale, and/or otherwise dispose of the Original Software
(or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on
the date Initial Developer first distributes or otherwise makes the
Original Software available to a third party under the terms of this
License.

(d) Notwithstanding Section 2.1(b) above, no patent license is
granted: (1) for code that You delete from the Original Software, or
(2) for infringements caused by: (i) the modification of the
Original Software, or (ii) the combination of the Original Software
with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject
to third party intellectual property claims, each Contributor hereby
grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or
trademark) Licensable by Contributor to use, reproduce, modify,
display, perform, sublicense and distribute the Modifications
created by such Contributor (or portions thereof), either on an
unmodified basis, with other Modifications, as Covered Software
and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling
of Modifications made by that Contributor either alone and/or in
combination with its Contributor Version (or portions of such
combination), to make, use, sell, offer for sale, have made, and/or
otherwise dispose of: (1) Modifications made by that Contributor (or
portions thereof); and (2) the combination of Modifications made by
that Contributor with its Contributor Version (or portions of such
combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective
on the date Contributor first distributes or otherwise makes the
Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is
granted: (1) for any code that Contributor has deleted from the
Contributor Version; (2) for infringements caused by: (i) third
party modifications of Contributor Version, or (ii) the combination
```

(continues on next page)

(continued from previous page)

```
  of Modifications made by that Contributor with other software
  (except as part of the Contributor Version) or other devices; or (3)
  under Patent Claims infringed by Covered Software in the absence of
  Modifications made by that Contributor.

3. Distribution Obligations.

  3.1. Availability of Source Code.

  Any Covered Software that You distribute or otherwise make available
  in Executable form must also be made available in Source Code form
  and that Source Code form must be distributed only under the terms
  of this License. You must include a copy of this License with every
  copy of the Source Code form of the Covered Software You distribute
  or otherwise make available. You must inform recipients of any such
  Covered Software in Executable form as to how they can obtain such
  Covered Software in Source Code form in a reasonable manner on or
  through a medium customarily used for software exchange.

  3.2. Modifications.

  The Modifications that You create or to which You contribute are
  governed by the terms of this License. You represent that You
  believe Your Modifications are Your original creation(s) and/or You
  have sufficient rights to grant the rights conveyed by this License.

  3.3. Required Notices.

  You must include a notice in each of Your Modifications that
  identifies You as the Contributor of the Modification. You may not
  remove or alter any copyright, patent or trademark notices contained
  within the Covered Software, or any notices of licensing or any
  descriptive text giving attribution to any Contributor or the
  Initial Developer.

  3.4. Application of Additional Terms.

  You may not offer or impose any terms on any Covered Software in
  Source Code form that alters or restricts the applicable version of
  this License or the recipients' rights hereunder. You may choose to
  offer, and to charge a fee for, warranty, support, indemnity or
  liability obligations to one or more recipients of Covered Software.
  However, you may do so only on Your own behalf, and not on behalf of
  the Initial Developer or any Contributor. You must make it
  absolutely clear that any such warranty, support, indemnity or
  liability obligation is offered by You alone, and You hereby agree
  to indemnify the Initial Developer and every Contributor for any
  liability incurred by the Initial Developer or such Contributor as a
  result of warranty, support, indemnity or liability terms You offer.

  3.5. Distribution of Executable Versions.
```

(continues on next page)

(continued from previous page)

```
  You may distribute the Executable form of the Covered Software under
  the terms of this License or under the terms of a license of Your
  choice, which may contain terms different from this License,
  provided that You are in compliance with the terms of this License
  and that the license for the Executable form does not attempt to
  limit or alter the recipient's rights in the Source Code form from
  the rights set forth in this License. If You distribute the Covered
  Software in Executable form under a different license, You must make
  it absolutely clear that any terms which differ from this License
  are offered by You alone, not by the Initial Developer or
  Contributor. You hereby agree to indemnify the Initial Developer and
  every Contributor for any liability incurred by the Initial
  Developer or such Contributor as a result of any such terms You offer.

  3.6. Larger Works.

  You may create a Larger Work by combining Covered Software with
  other code not governed by the terms of this License and distribute
  the Larger Work as a single product. In such a case, You must make
  sure the requirements of this License are fulfilled for the Covered
  Software.

4. Versions of the License.

  4.1. New Versions.

  Oracle is the initial license steward and may publish revised and/or
  new versions of this License from time to time. Each version will be
  given a distinguishing version number. Except as provided in Section
  4.3, no one other than the license steward has the right to modify
  this License.

  4.2. Effect of New Versions.

  You may always continue to use, distribute or otherwise make the
  Covered Software available under the terms of the version of the
  License under which You originally received the Covered Software. If
  the Initial Developer includes a notice in the Original Software
  prohibiting it from being distributed or otherwise made available
  under any subsequent version of the License, You must distribute and
  make the Covered Software available under the terms of the version
  of the License under which You originally received the Covered
  Software. Otherwise, You may also choose to use, distribute or
  otherwise make the Covered Software available under the terms of any
  subsequent version of the License published by the license steward.

  4.3. Modified Versions.

  When You are an Initial Developer and You want to create a new
  license for Your Original Software, You may create and use a
```

(continues on next page)

(continued from previous page)

```
  modified version of this License if You: (a) rename the license and
  remove any references to the name of the license steward (except to
  note that the license differs from this License); and (b) otherwise
  make it clear that the license contains terms which differ from this
  License.

5. DISCLAIMER OF WARRANTY.

  COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS,
  WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
  INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE
  IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR
  NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF
  THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE
  DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY
  OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING,
  REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN
  ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS
  AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

  6.1. This License and the rights granted hereunder will terminate
  automatically if You fail to comply with terms herein and fail to
  cure such breach within 30 days of becoming aware of the breach.
  Provisions which, by their nature, must remain in effect beyond the
  termination of this License shall survive.

  6.2. If You assert a patent infringement claim (excluding
  declaratory judgment actions) against Initial Developer or a
  Contributor (the Initial Developer or Contributor against whom You
  assert such claim is referred to as "Participant") alleging that the
  Participant Software (meaning the Contributor Version where the
  Participant is a Contributor or the Original Software where the
  Participant is the Initial Developer) directly or indirectly
  infringes any patent, then any and all rights granted directly or
  indirectly to You by such Participant, the Initial Developer (if the
  Initial Developer is not the Participant) and all Contributors under
  Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice
  from Participant terminate prospectively and automatically at the
  expiration of such 60 day notice period, unless if within such 60
  day period You withdraw Your claim with respect to the Participant
  Software against such Participant either unilaterally or pursuant to
  a written agreement with Participant.

  6.3. If You assert a patent infringement claim against Participant
  alleging that the Participant Software directly or indirectly
  infringes any patent where such claim is resolved (such as by
  license or settlement) prior to the initiation of patent
  infringement litigation, then the reasonable value of the licenses
  granted by such Participant under Sections 2.1 or 2.2 shall be taken
```

(continues on next page)

(continued from previous page)

```
  into account in determining the amount or value of any payment or
  license.

  6.4. In the event of termination under Sections 6.1 or 6.2 above,
  all end user licenses that have been validly granted by You or any
  distributor hereunder prior to termination (excluding licenses
  granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

  UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT
  (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE
  INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF
  COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE
  TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR
  CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT
  LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER
  FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR
  LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE
  POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT
  APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH
  PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH
  LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR
  LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION
  AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

  The Covered Software is a "commercial item," as that term is defined
  in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer
  software" (as that term is defined at 48 C.F.R. §
  252.227-7014(a)(1)) and "commercial computer software documentation"
  as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent
  with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4
  (June 1995), all U.S. Government End Users acquire Covered Software
  with only those rights set forth herein. This U.S. Government Rights
  clause is in lieu of, and supersedes, any other FAR, DFAR, or other
  clause or provision that addresses Government rights in computer
  software under this License.

9. MISCELLANEOUS.

  This License represents the complete agreement concerning subject
  matter hereof. If any provision of this License is held to be
  unenforceable, such provision shall be reformed only to the extent
  necessary to make it enforceable. This License shall be governed by
  the law of the jurisdiction specified in a notice contained within
  the Original Software (except to the extent applicable law, if any,
  provides otherwise), excluding such jurisdiction's conflict-of-law
  provisions. Any litigation relating to this License shall be subject
  to the jurisdiction of the courts located in the jurisdiction and
```

(continues on next page)

(continued from previous page)

```
  venue specified in a notice contained within the Original Software,
  with the losing party responsible for costs, including, without
  limitation, court costs and reasonable attorneys' fees and expenses.
  The application of the United Nations Convention on Contracts for
  the International Sale of Goods is expressly excluded. Any law or
  regulation which provides that the language of a contract shall be
  construed against the drafter shall not apply to this License. You
  agree that You alone are responsible for compliance with the United
  States export administration regulations (and the export control
  laws and regulation of any other countries) when You use, distribute
  or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

  As between Initial Developer and the Contributors, each party is
  responsible for claims and damages arising, directly or indirectly,
  out of its utilization of rights under this License and You agree to
  work with Initial Developer and Contributors to distribute such
  responsibility on an equitable basis. Nothing herein is intended or
  shall be deemed to constitute any admission of liability.
```

# Chapter 14

# Appendices

## 14.1 Appendix A: AUTOSAR Integration Example

This guide walks you through an example of integrating *Connext Micro* into your AUTOSAR Classic Platform workspace as a Complex Device Driver (CDD). It is written for platform integrators with experience in AUTOSAR software, network configuration, and basic DDS concepts.

We'll cover how to get DDS communication working in your AUTOSAR basic software (BSW), understanding the architecture and design, and customizing DDS for your application needs. When finished, you should be able to the following:

- Electronic Control Unit (ECU) boots and acquires IP address;
- DDS CDD initializes successfully;
- *Admin Console* discovers the ECU *DomainParticipant*;
- Bidirectional data exchange verified.

This guide does NOT cover production deployment (beyond AUTOSAR integration), application-specific requirements, or platform-specific optimizations.

### 14.1.1 How DDS integrates with AUTOSAR

Before diving into configuration, let's understand how DDS integrates into the AUTOSAR Classic Platform.

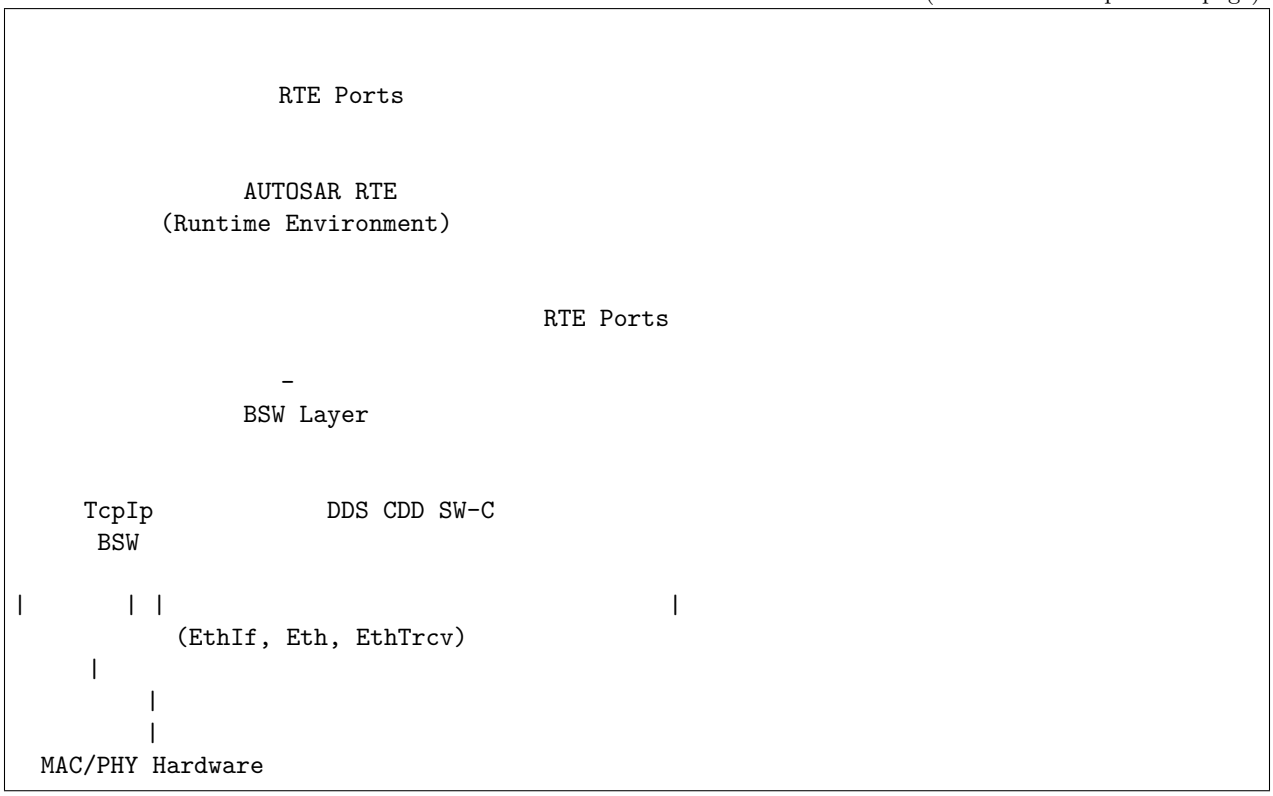
DDS is integrated as a CDD Software Component (SW-C) that acts as a proxy between Application SW-Cs and the network. DDS *Topics* map to RTE port interfaces, enabling Application SW-Cs to publish and subscribe to DDS data through standard RTE API calls.

The following diagram outlines the data flow through a DDS-integrated system:

```
       Application SW-C
          (Your App)
```

(continues on next page)

(continued from previous page)




We can summarize the data flow as follows:

**Publishing:** When publishing data, the application SW-C writes to an RTE port, which triggers the DDS CDD runnable. The DDS CDD then publishes the data via the TcpIp stack to the network.

**Subscribing:** When subscribing to data, the TcpIp stack receives data from the network, and the DDS Middleware (*Connext Micro*) processes it by storing samples, applying QoS policies, and managing reliability. The DDS CDD periodically polls the *DataReader* to check for new data, converts it, and writes it to the RTE port. Finally, the application SW-C reads the data from the RTE port.

From the perspective of the application SW-C, DDS communication looks like standard AUTOSAR inter-SW-C communication through RTE ports.

---

**Note:** DDS data processing can alternatively be triggered by event-driven callbacks using WaitSets or other *Connext Micro* mechanisms. This reference implementation uses periodic polling via cyclic tasks for simplicity.

---

### 14.1.2 Integration steps

Before starting, you must have the following hardware and software:

- A Target ECU with Ethernet MAC (Media Access Control) and PHY (Physical Layer Transceiver)
- Ethernet cable and network switch
- Windows or Linux PC for *Admin Console*
- AUTOSAR BSW with operational TcpIp stack
- Run Time Environment (RTE) generator and configuration tools
- Build toolchain for target platform
- RTI *Admin Console* installed

#### Reference implementation

RTI provides a complete reference implementation of AUTOSAR integration, consisting of:

- A reference DDS-XML configuration (`dds_system.xml`);
- A complete DDS CDD: SW-C template (reference ARXML), adapter, and DDS implementation;
- An example application SW-C (reference ARXML).

**Attention:** The `dds_system.xml` file is the "source of truth" for DDS configuration. Network settings (such as IP addresses and domain IDs) and QoS profiles defined here *must* match your AUTOSAR BSW configuration.

The following will be used as-is:

- DDS CDD source code and SW-C ARXML
- Example application SW-C ARXML
- State machine initialization sequence
- Example data types (initially)

You must configure the following for your specific platform:

- Network settings (IP address)
- BSW integration (TcpIp, OS, BswM)
- Reference SW-C integration (import reference ARXML files, no modification needed)
- Build system integration

### Configuring the AUTOSAR BSW

Your AUTOSAR BSW is configured in `MCU_DDS_CDD/config/dds_system.xml`. Before configuring, review `dds_system.xml` to understand some key parts of the reference configuration:

- Network settings: IP address (e.g., 192.168.1.100), netmask, and multicast addresses.
- Domain ID: Found in `<domain name="myDomain" domain_id="0">`.
- QoS profiles: `DefaultProfile`, `CddProfile`, and `PcProfile`.
- Topics: `MCUtoPC` and `PCtoMCU`.

Your AUTOSAR BSW configuration must match these DDS-XML settings.

This section contains ALL required BSW configuration. The integration will not work without these settings.

### TcpIp BSW Configuration

1. Ensure the socket owner `GetSocket` function matches the one configured by `DDSCDD_SOCKET_OWNER_GET_SOCKET` macro:

```
<SOCKET-CONNECTION-BUNDLE>
    <SHORT-NAME>DdsSocketOwner</SHORT-NAME>
    <SOCKET-PROTOCOL>UDP</SOCKET-PROTOCOL>
</SOCKET-CONNECTION-BUNDLE>
```

2. Register these callbacks in TcpIp using the vendor tool to update the ARXML configuration:

```
<SOCKET-OWNER>
   <SHORT-NAME>DdsSocketOwner</SHORT-NAME>
   <RX-INDICATION-FUNCTION>DDS_CDD_RxIndication</RX-INDICATION-FUNCTION>
   <LOCAL-IP-ADDR-ASSIGNMENT-CHG-FUNCTION>DDS_CDD_LocalIpAddrAssignmentChg</LOCAL-
↪IP-ADDR-ASSIGNMENT-CHG-FUNCTION>
</SOCKET-OWNER>
```

   - `DDS_CDD_RxIndication`: called when UDP data received.
   - `DDS_CDD_LocalIpAddrAssignmentChg`: called when IP address assigned.
   - `DDS_CDD_TcpIpEvent`: called when a TcpIp event occurs.

3. Use callback prototypes that match the AUTOSAR TcpIp API types:

```
void DDS_CDD_LocalIpAddrAssignmentChg(
   TcpIp_LocalAddrIdType LocalAddrId,
   TcpIp_IpAddrStateType State);

void DDS_CDD_RxIndication(
   TcpIp_SocketIdType SocketId,
   P2CONST(TcpIp_SockAddrType, AUTOMATIC, TCPIP_APPL_DATA) RemoteAddrPtr,
   P2VAR(uint8, AUTOMATIC, TCPIP_APPL_DATA) BufPtr,
```

(continues on next page)

(continued from previous page)

```
   uint16 Length);

void DDS_CDD_TcpIpEvent(
   TcpIp_SocketIdType SocketId,
   TcpIp_EventType Event);
```

The callbacks must produce the following behavior:

- `DDS_CDD_LocalIpAddrAssignmentChg` gates startup and should trigger `Rte_Start()` only when the target unicast address is assigned.
- `DDS_CDD_RxIndication` forwards received UDP payloads to the DDS NETIO receive path.
- `DDS_CDD_TcpIpEvent` handles socket lifecycle events (or can log and ignore unused events).

---

**Note:** The callback names configured in ARXML must exactly match the compiled function symbols in your integration code.

---

4. Ensure TcpIp stack configuration allocates sufficient sockets beyond any existing usage. DDS requires at least 3 additional UDP sockets (2 for unicast, 1 for multicast, as configured in `dds_system.xml`).
5. Check `MCU_DDS_CDD/config/dds_system.xml` for the reference IP configuration. Look in the `CddProfile` QoS profile under `<transport_builtin><udpv4><interface_table>` for the default reference configuration (`192.168.1.100` with netmask `255.255.255.0`).
6. Configure the TcpIp BSW to match `dds_system.xml`:

   **Unicast IP Configuration:**

```
<LOCAL-ADDRESS-CONFIG>
   <SHORT-NAME>LocalAddress_ECU</SHORT-NAME>
   <LOCAL-ADDRESS-ID>0</LOCAL-ADDRESS-ID>  <!-- MUST be 0 for unicast -->
   <IP-ADDRESS-ASSIGNMENT-METHOD>STATIC</IP-ADDRESS-ASSIGNMENT-METHOD>
   <IP-ADDRESS>192.168.1.100</IP-ADDRESS>  <!-- MUST MATCH dds_system.xml␣
↪CddProfile -->
   <NETWORK-MASK>255.255.255.0</NETWORK-MASK>
   <DEFAULT-GATEWAY>192.168.1.1</DEFAULT-GATEWAY>
</LOCAL-ADDRESS-CONFIG>
```

---

**Note:** The IP address, netmask, and interface settings *must match* what's configured in `dds_system.xml` under the `CddProfile`.

---

**Multicast IP Configuration:**

Check `dds_system.xml` for multicast settings. Look in `<discovery><initial_peers>`, which

includes `239.255.0.1`. Look in `<multicast_receive_addresses>`, which defines `udpv4://239.255.0.1`

```
<LOCAL-ADDRESS-CONFIG>
   <SHORT-NAME>MulticastAddress_DDS</SHORT-NAME>
   <LOCAL-ADDRESS-ID>1</LOCAL-ADDRESS-ID>
   <IP-ADDRESS-ASSIGNMENT-METHOD>STATIC</IP-ADDRESS-ASSIGNMENT-METHOD>
   <IP-ADDRESS>239.255.0.1</IP-ADDRESS>  <!-- MUST MATCH dds_system.xml -->
</LOCAL-ADDRESS-CONFIG>
```

### OS configuration

The DDS CDD requires a single OS resource for internal synchronization and state management. This resource is used for protecting critical sections in the adapter layer and ensuring thread safety.

```
<!-- Example: Resource 001 -->
<RESOURCE>
    <SHORT-NAME>DDS_Resource001</SHORT-NAME>
    <RESOURCE-PROPERTY>STANDARD</RESOURCE-PROPERTY>
</RESOURCE>
```

The DDS CDD also requires a task, configured as follows:

```
<TASK>
    <SHORT-NAME>Task_DdsCdd</SHORT-NAME>
    <PRIORITY>8</PRIORITY>  <!-- Adjust based on your system -->
    <SCHEDULE>FULL</SCHEDULE>
    <ACTIVATION>1</ACTIVATION>
    <AUTOSTART>true</AUTOSTART>
    <STACK-SIZE>13312</STACK-SIZE>  <!-- See note below -->
    <EVENT-REF DEST="EVENT">Event_DdsCddRun</EVENT-REF>
    <EVENT-REF DEST="EVENT">Event_DdsCddRead</EVENT-REF>
    <EVENT-REF DEST="EVENT">Event_DdsCddWrite</EVENT-REF>
</TASK>
```

- Stack size: Allocate an additional 10 KB of stack space for DDS CDD runnables. If reusing an existing task, increase its stack size by 10 KB. If creating a new task, ensure at least 10 KB is available. This is an estimate, and actual requirements may vary based on your application and DDS configuration.
- Task priority: DDS CDD runnables should have higher priority than application SW-Cs but lower priority than critical BSW tasks (such as the ethernet driver or TcpIp stack). This ensures timely network communication without blocking critical system functions.
- Runnable mapping: DdsCdd runnables (e.g., `DdsCddStart`, `DdsCddRun`, `DdsCddRxIndication`, `DdsCddProcessData`, `DdsCddRead_PCtoMCU`, or `DdsCddWrite_MCUtoPC`) must be mapped to `Task_DdsCdd` in your RTE configuration. You will configure trigger conditions for these runnables in *Configuring RTE*.

### Basic Software Mode Manager (BswM) configuration

> **Attention:** `Rte_Start()` must *not* be called from a BswM action list. It must be called from the `DDS_CDD_LocalIpAddrAssignmentChg` callback, only after the IP address has been assigned and the Ethernet link is up.

DDS initialization sends discovery ping messages to other participants. If `Rte_Start()` is called before the TcpIp stack has assigned an IP address and the link is up, DDS initialization will fail.

The correct startup sequence is:

1. BswM invokes `TcpIp_Init()` as normal.
2. TcpIp calls `DDS_CDD_LocalIpAddrAssignmentChg` when the IP address is assigned.
3. The callback calls `Rte_Start()`, which triggers `DdsCddStart` (InitEvent), where all DDS initialization takes place.

```
void DDS_CDD_LocalIpAddrAssignmentChg(
    TcpIp_LocalAddrIdType LocalAddrId,
    TcpIp_IpAddrStateType State)
{
    if (LocalAddrId == TCPIP_LOCALADDRID_UNICAST &&
        State == TCPIP_IPADDR_STATE_ASSIGNED)
    {
        Rte_Start();
    }
}
```

---

**Note:** This guide documents one valid initialization strategy: starting RTE after IP assignment, then performing DDS initialization in `DdsCddStart`. Other strategies are possible. For example, RTE can be started in the standard BswM sequence, and DDS initialization can be deferred to a periodic runnable that monitors network readiness.

---

### Ethernet driver configuration

The DDS CDD interfaces with TcpIp stack. Ethernet driver configuration is the BSW integrator's responsibility according to their hardware requirements.

---

**Note:** Ensure Ethernet MAC duplex mode matches PHY duplex mode to avoid packet loss.

---

### Configuring RTE

1. Import the provided ARXML files containing:
   - DdsCdd ARXML: DDS CDD SW-C definition, port interfaces, port prototypes, and runnable definitions.
   - ExampleAsrApp ARXML: Example application SW-C and example sender-receiver interfaces.
2. Connect DDS CDD ports to application SW-C, as shown in the following diagram:

   code-block:: none

   Application SW-C DDS CDD SW-C P_DataOut → R_MCUtoPC (Publishing) R_DataIn ← S_PCtoMCU (Subscribing)
3. Before generating RTE code, map the DDS CDD runnables to the task you configured in OS section (`Task_DdsCdd`):
4. Run your RTE generator to create the following:
   - `Rte.c` / `Rte.h`
   - `Rte_DdsCdd.h`
   - `Rte_ExampleAsrApp.h`
5. In DdsCdd ARXML, configure trigger conditions for the following runnables:

| Runnable | Trigger Condition | Purpose |
|---|---|---|
| `DdsCddStart()` | InitEvent | Initialize DDS CDD at startup |
| `DdsCddRun()` | TimingEvent (10ms) | DDS QoS maintenance (discovery, liveliness, etc.) |
| `DdsCddProcessData()` | Inter-runnable Trigger (called by DdsCddRxIndication) | Process the DDS data. |
| `DdsCddRxIndication()` | No trigger defined (called by connext library) | Triggers the Inter-runnable trigger for processing received data. |
| `DdsCddRead_PCtoMCU()` | TimingEvent (100ms) | Poll for incoming DDS data |
| `DdsCddWrite_MCUtoPC()` | DataReceivedEvent | Publish data when app writes to RTE port |

**Warning:** If DDS initialization is executed inside a periodic runnable, the first activation can exceed the nominal runnable period (for example, a 10 ms runnable may run longer during initialization). Account for this during OS scheduling and timing analysis (watchdog, deadline monitoring, and task load budget).

**Note:** The RTE automatically generates the required OS alarms and events based on these trigger condition configurations. You do not manually configure alarms or events; the RTE

generator creates them from the trigger conditions and task mapping you specify.

6. After RTE code generation, implement the generated DdsCdd runnable bodies as thin wrappers to Layer 2 adapter functions.

   Use this pattern:

```
FUNC(void, DdsCdd_CODE) DdsCddStart(void)
{
   DdsCdd_Adapter_Init();
}

FUNC(void, DdsCdd_CODE) DdsCddRun(void)
{
   DdsCdd_Adapter_Run();
}

FUNC(void, DdsCdd_CODE) DdsCddProcessData(void)
{
   NETIO_Autosar_udp_receive_callback();
}

FUNC(void, DdsCdd_CODE) DdsCddRead_PCtoMCU(void)
{
   DdsCdd_Adapter_Read_PCtoMCU();
}

FUNC(void, DdsCdd_CODE) DdsCddWrite_MCUtoPC(void)
{
   DdsCdd_Adapter_Write_MCUtoPC();
}
```

   Implementation checklist:

   - Keep runnable bodies minimal; put DDS logic in adapter/implementation layers.
   - Modify only the user-editable sections in generated template files.
   - Keep trigger-condition semantics unchanged (InitEvent, TimingEvent, DataReceivedEvent, internal trigger).
   - Re-run RTE generation after ARXML changes and re-apply user code in the generated implementation blocks.

   For the architectural rationale behind this split, see *Appendix B: AUTOSAR Architecture Example*.

### Integrating source code

1. Add the following source files to your build:
   - *Connext Micro*:
     - All source files from `$(RTI_DDS_MICRO)/src/`
   - DDS generated code:
     - `MCU_DDS_CDD/dds_gen/exampleType.c`
     - `MCU_DDS_CDD/dds_gen/exampleTypePlugin.c`
     - `MCU_DDS_CDD/dds_gen/exampleTypeSupport.c`
   - DDS CDD implementation:
     - `MCU_DDS_CDD/adaptation/dds_cdd_adapter.c`
     - `MCU_DDS_CDD/dds_impl/dds_impl.c`
     - `MCU_DDS_CDD/dds_cdd/DdsCdd.c`
   - Example application:
     - `MCU_EXAMPLE_ASR_APP/ExampleAsrApp.c`
2. Configure the necessary include paths:

```
-I$(RTI_DDS_MICRO)/include
-I$(RTI_DDS_MICRO)/include/autosar
-I$(PROJECT)/MCU_DDS_CDD/config
-I$(PROJECT)/MCU_DDS_CDD/dds_gen
-I$(PROJECT)/MCU_DDS_CDD/adaptation
-I$(PROJECT)/MCU_DDS_CDD/dds_impl
-I$(PROJECT)/MCU_EXAMPLE_ASR_APP
```

   Do not include `MCU_DDS_CDD/dds_cdd/DdsCdd`; these are the Contract Phase Headers and are already included via RTE generated code.

3. Configure the following preprocessor definitions:

```
#define __autosar__                       // AUTOSAR platform
#define RTI_ENDIAN_LITTLE                 // Or RTI_ENDIAN_BIG (based on target)
#define NETIO_CONFIG_ENABLE_MULTICAST 1   // Multicast configured in dds_system.xml
#define OSAPI_ENABLE_LOG 0                // Disable logging
#define OSAPI_ENABLE_TRACE 0              // Disable logging
```

4. Configure memory. The DDS CDD requires static memory allocation; the default heap size is 240KB (the baseline for 2 topics with small samples).

```
#define DDSCDD_HEAP_AREA_1_SIZE (240 * 1024)  // 240KB
```

   Memory is used for DDS entities (participant, topics, readers, writers), discovery protocol data, send/receive buffers, QoS policy storage.

**Note:** The default 240KB is for a baseline system (2 topics with small samples). See *Customizing for your application* for sizing guidelines.

5. Configure DDS entities using APPGEN. DDS entities (*DomainParticipants*, *Topics*, *DataWriters*, and *DataReaders*) are defined in `dds_system.xml` and loaded at runtime by Layer 3 (`dds_impl.c`) using the Application Generation (APPGEN) plugin. APPGEN is the standard *Connext Micro* mechanism for XML-driven entity creation.

   During `DdsImpl_CreateEntities()`, the APPGEN plugin is registered with the DDS factory and the participant is created from the XML profile:

```
/* Register the APPGEN plugin to enable XML-based entity creation */
struct APPGEN_FactoryProperty model_xml = APPGEN_FactoryProperty_INITIALIZER;
model_xml._model = APPGEN_get_library_seq();
if (!APPGEN_Factory_register(registry, &model_xml)) {
   /* handle error */
}

/* Create participant from the profile defined in dds_system.xml */
application->participant =
   DDS_DomainParticipantFactory_create_participant_from_config(
            factory,
            "AppLibrary::MCU_DDS_CDD");
```

   The participant name (`"AppLibrary::MCU_DDS_CDD"`) must match the `<domain_participant_library name="AppLibrary">` and `<domain_participant name="MCU_DDS_CDD">` entries in your `dds_system.xml`. All child entities (publishers, subscribers, writers, readers) are created automatically from the XML.

   The participant is created in **disabled** state. It is enabled separately in `DdsImpl_EnableEntities()` once the initialization state machine reaches `EntitiesCreated`.

   **Note:** APPGEN requires the following additional includes in `dds_impl.c`:

```
#include "app_gen/app_gen.h"
#include "app_gen/app_gen_plugin.h"
```

   These headers are provided by *Connext Micro* and should not be modified.

### Building and flashing

We recommend a full rebuild for your first integration; the build should complete without errors. Verify that no linker errors occur related to memory size.

Use your standard flashing procedure and verify that sufficient flash space is available.

### Validating integration

Check the following:

- The ECU boots without errors;
- The TcpIp stack initializes and assigns IP;
- The DDS CDD state machine reaches ENTITIES_ENABLED;
- There are no hardfaults or exceptions.

For debugging, we recommend the following:

- Check the DDS CDD state via the debugger with the `DdsCdd_State` variable.
- Verify the IP assignment with the `DdsCdd_IpAddressAssigned` flag.
- Monitor initialization sequence timing.

To test network connectivity, run the following from the command line:

```
ping 192.168.1.100  # ECU IP address
```

The ping response should succeed.

To validate DDS discovery, set up *Admin Console*:

1. Launch *Admin Console* on your PC.
2. Set the PC IP address to 192.168.1.10 (same subnet as the ECU).
3. Load the QoS profile `admin_console_qos.xml`.
4. Set the Domain ID to `0` (configured in `dds_system.xml` under `<domain name="myDomain" domain_id="0">`).

For detailed instructions on how to set up *Admin Console*, see the Admin Console User's Manual.

Verify that discovery has succeeded by confirming the following:

- The ECU *DomainParticipant* appears in *Participants* view;
- The *DataWriter* "MCUtoPC" is visible;
- The *DataReader* "PCtoMCU" is visible.

To test PC-to-ECU communication:

1. In *Admin Console*, publish a sample to the "PCtoMCU" topic;
2. Verify that the ECU receives data (e.g., check via the debugger or application behavior).

Test ECU-to-PC communication:

1. Trigger a sample publication from the ECU to the application;
2. In *Admin Console*, subscribe to the "MCUtoPC" topic;
3. Verify that samples appear with the correct data values.

At this point, you should have verified that:

- The ECU boots and initializes DDS successfully;
- Network connectivity functions;
- DDS discovery succeeded;
- PC-to-ECU communication succeeded;
- ECU-to-PC communication succeeded;
- Data integrity was confirmed.

Congratulations! You have successfully integrated DDS into your AUTOSAR workspace.

### 14.1.3 Understanding the architecture

Now that you have DDS working, let's review what you've integrated.

The DDS CDD uses a three-layer design that separates AUTOSAR concerns from DDS logic:

```
Layer 1: RTE Template (DdsCdd.c)
• AUTOSAR runnables called by RTE
• Generated from ARXML
• Minimal logic - delegates to adapter



Layer 2: Adapter (dds_cdd_adapter.c/h)
• Type conversion (AUTOSAR  DDS)
• TcpIp integration and callbacks
• Initialization state machine
• AUTOSAR resource management



Layer 3: DDS Implementation (dds_impl.c/h)
• Pure DDS operations
• Entity creation and management
• Read/write operations
• No AUTOSAR dependencies
```

This approach ensures that Layer 1 changes only when ARXML changes, Layer 2 bridges AUTOSAR and DDS, and Layer 3 is portable and independently testable.

DDS initialization proceeds through the following states to ensure correct sequencing:

```
TcpIpNotReady → TcpIpReady → SystemPropertiesSet → EntitiesCreated → EntitiesEnabled
```

When publishing, the application writes to the RTE, which triggers a runnable in the DDS CDD. The adapter then converts the data types and the DDS layer writes the data to the network.

When subscribing, network data is first received by the TcpIp stack. The DDS middleware stores the data in a queue and a periodic runnable polls the *DataReader* for new data. The adapter converts the data types and the RTE delivers the data to the application SW-C.

For more detailed architecture information, see *Appendix B: AUTOSAR Architecture Example*. We recommend reading the Architecture section now, or returning to it after finishing with this guide.

### 14.1.4 Customizing for your application

#### Add a new data type and Topic

1. Define a type in DDS-XML:

   Edit `dds_system.xml` as shown below:

   ```
   <types>
       <struct name="VehicleSpeed">
           <member name="timestamp" type="ulonglong"/>
           <member name="speed_kph" type="float"/>
           <member name="vehicle_id" type="long"/>
       </struct>
   </types>
   ```

2. Generate type support code. See *Application Generation Using XML* for more information.

3. Add a topic and endpoints to DDS-XML, as shown below:

   ```
   <topic name="VehicleSpeedTopic" register_type_ref="VehicleSpeed">
       <topic_qos base_name="QosLibrary::DefaultProfile"/>
   </topic>
   ...
   <data_writer name="VehicleSpeedWriter" topic_ref="VehicleSpeedTopic"/>
   ```

4. Update ARXML.

   Manually create (or use arcgen/MAG generated arxml) the ARXML elements to integrate the new DDS topic into your application, as follows:

   - Create a data type compatible with the DDS type (e.g., `VehicleSpeed_t`).
   - Create a sender-receiver port interface using the new data type.
   - Create port prototypes in DdsCdd SW-C. For publishing topics, create a receive port (`R_VehicleSpeed`). For subscribing topics, create a send port (`S_VehicleSpeed`).
   - Create one runnable per new port prototype (e.g., `Write_VehicleSpeed()` for publishing).

- Map each runnable to its corresponding port prototype.
- Set trigger conditions for runnables. Publishing runnables should be triggered by `DataReceivedEvent` on input port. Subscribing runnables should be triggered by `TimingEvent` (periodic polling).
- Repeat the steps above for your application SW-C. Create send port prototypes for topics your application publishes, create receive port prototypes for topics your application subscribes to, and create runnables to handle data exchange.
- Wire your application's SW-C output ports to DdsCdd input ports (publishing path). Wire DdsCdd output ports to your application's SW-C input ports (subscribing path).
- Assign all new runnables to the appropriate OS task (e.g., Task_DdsCdd).

5. Adjust static heap memory.

   Memory per topic depends on the sample size (user data payload), History depth (`KEEP_LAST` depth, `KEEP_ALL` not recommended), the number of remote *DomainParticipants*/*DataReaders*/*DataWriters*, etc. For more details, refer to *Memory Management*.

   For example:

```
#define DDSCDD_HEAP_AREA_1_SIZE  (400 * 1024)  // 400KB
```

6. Add adapter functions. In `dds_cdd_adapter.c`:

```
void DdsCdd_Adapter_Write_VehicleSpeed(void)
{
    VehicleSpeed_t rteData;
    VehicleSpeed ddsData;

    Rte_Read_R_VehicleSpeed(&rteData);

    // Type conversion
    ddsData.timestamp = rteData.timestamp;
    ddsData.speed_kph = rteData.speed_kph;
    ddsData.vehicle_id = rteData.vehicle_id;

    DdsImpl_Write_VehicleSpeed(&ddsData);
}
```

7. Add implementation functions. In `dds_impl.c`:

```
DDS_ReturnCode_t DdsImpl_Write_VehicleSpeed(const VehicleSpeed* data)
{
    return VehicleSpeed_DataWriter_write(g_VehicleSpeedWriter, data, &DDS_HANDLE_
↪NIL);
}
```

8. Generate OS code from your OS ARXML configuration.
9. Generate RTE code from your RTE/SWC ARXML configuration.
10. Generate SWC template code for DdsCdd SW-C and all application SW-Cs.

11. Implement DdsCdd template runnables. Fill in the generated DdsCdd runnable stubs and call the corresponding adapter functions (e.g., `DdsCdd_Adapter_Write_VehicleSpeed()`). For example:

```
void DdsCdd_Write_VehicleSpeed(void)
{
      DdsCdd_Adapter_Write_VehicleSpeed();
}
```

12. Compile all generated code and implementation files. Verify there are no compilation or linker errors.

13. Validate the following:

    - Check *Connext Micro* static heap usage;
    - Use *Admin Console* to verify the new topic;
    - Test bidirectional communication;
    - Verify data integrity.

### Change QoS from Reliable to Best-Effort

This is a possible use case for high-frequency sensor data where occasional loss is acceptable.

Edit `dds_system.xml` as follows:

```
<qos_profile name="BestEffortProfile">
    <datawriter_qos>
        <reliability>
            <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
        </reliability>
        <history>
            <kind>KEEP_LAST_HISTORY_QOS</kind>
            <depth>1</depth>
        </history>
    </datawriter_qos>
    <datareader_qos>
        <reliability>
            <kind>BEST_EFFORT_RELIABILITY_QOS</kind>
        </reliability>
        <history>
            <kind>KEEP_LAST_HISTORY_QOS</kind>
            <depth>1</depth>
        </history>
    </datareader_qos>
</qos_profile>

<topic name="SensorData" register_type_ref="SensorType">
    <topic_qos base_name="QosLibrary::BestEffortProfile"/>
</topic>
```

This configuration benefits from lower latency, reduced network overhead, and no reliability protocol overhead. However, packets may be lost and delivery is not guaranteed.

---

**Note:** QoS changes in `dds_system.xml` only require regenerating DDS type support code. No changes are needed in the adapter layer (`dds_cdd_adapter.c/h`), AUTOSAR template layer (`DdsCdd.c`), or the application SW-Cs.

---

#### Tune timing for your application

To adjust runnable periods, use the following guidelines:

DdsCddRun (DDS maintenance):

- Default: 10ms
- Faster (5ms): Lower discovery time, higher CPU usage.
- Slower (20ms): Higher latency for QoS protocols.

DdsCddRead (polling for data):

- Default: 100ms

  Adjust based on expected data rate, such as 50ms for higher update rate.

---

**Note:** Timing adjustments only require runnable configuration changes in DdsCdd ARXML. No modifications are needed in the adapter layer (`dds_cdd_adapter.c/h`) or DDS implementation layer (`dds_impl.c/h`).

---

### 14.1.5 Troubleshooting

This section describes how to troubleshoot some common integration issues.

#### DDS entities not created

Check the following:

- The IP address is assigned (was the TcpIp callback triggered?);
- The heap size is sufficient;
- 1 OS resource is configured as a STANDARD resource;
- The socket owner name matches the configuration;
- `Rte_Start()` is called from `DDS_CDD_LocalIpAddrAssignmentChg` only after the IP address is assigned (not from a BswM action list).

Debug:

```
// Check state machine
printf("DdsCdd_State: %d\n", DdsCdd_State);
printf("IP assigned: %d\n", DdsCdd_IpAddressAssigned);
```

#### No DDS discovery

Check the following:

- Network connectivity (does the ping test pass?);
- The domain ID matches between the ECU and PC;
- The firewall is not blocking UDP ports;
- Communicating on the same subnet (or routing configured).

Debug:

- Use *Wireshark* to capture RTPS packets
- Use *Admin Console* to check for discovery issues, QoS policy mismatch, etc.

#### Data not received

Check the following:

- Is the `DdsCddRead` runnable executing?
- QoS policies are compatible between the *DataWriter* and *DataReader*;
- Topic names match exactly;
- RTE ports are connected.

Debug:

- Use *Wireshark* to capture RTPS packets.
- Use *Admin Console* to check for topic data.

#### High CPU usage

Check the following:

- Is the `DdsCddRun` period too fast?
- Is network congestion causing retransmissions?
- Is the task priority too high?

Possible solutions:

- Increase `DdsCddRun` period to 20ms.
- Use `BEST_EFFORT` Reliability where appropriate.

- Reduce unnecessary data traffic.

#### Out of memory

Checkthe following:

- Is the heap size sufficient for your configuration?
- Are there too many topics?
- Is the History QoS policy depth too large?

Possible solutions:

- Increase `DDSCDD_HEAP_AREA_1_SIZE`.
- Reduce History depth.
- Use `KEEP_LAST` History.

### 14.1.6 API quick reference

AUTOSAR Layer (`Rte_DdsCdd_Type.h`):

- `Rte_Write_P_<DDS_topic>_<Type>()`: Write data to RTE port.
- `Rte_Read_R_<DDS_topic>_<Type>()`: Read data from RTE port.

Adapter Layer (`dds_cdd_adapter.h`):

- `DdsCdd_Adapter_Init()`: Initialize adapter.
- `DdsCdd_Adapter_Run()`: Cyclic execution (10ms).
- `DdsCdd_Adapter_Read_<DDS_topic>()`: Read from DDS.
- `DdsCdd_Adapter_Write_<DDS_topic>()`: Write to DDS.

DDS Implementation Layer (`dds_impl.h`):

- `DdsImpl_Init()`: Create DDS entities.
- `DdsImpl_Enable()`: Enable entities.
- `DdsImpl_Write_<DDS_topic>()`: Publish data.
- `DdsImpl_Read_<DDS_topic>()`: Read data.
- `DdsImpl_Run()`: Maintain DDS system.

### 14.1.7 Memory sizing

The baseline memory sizing comprises 240KB, which includes 2 topics, small samples (<100 bytes), `KEEP_LAST` History (depth 10), and 1 *DomainParticipant*.

When adjusting from this baseline, use the following rules of thumb:

- For each additional topic, add 40-60KB;
- For large samples (>1KB), add `sample_size * history_depth`;
- For discovery, add 10KB per remote *DomainParticipant*;

For a more conservative approach, start with the default 240KB, profile your actual memory usage on the target platform, add a 20-30% margin, and test the worst-case scenarios.

### 14.1.8 Related documentation

Refer to *Appendix B: AUTOSAR Architecture Example* for a comprehensive technical architecture of *Connext Micro* and AUTOSAR.

## 14.2 Appendix B: AUTOSAR Architecture Example

This section will help you understand the design of *Connext Micro*-AUTOSAR compatibility in depth, allowing you to modify the implementation for advanced use cases or explain the system in detail. It can also help with debugging complex integration issues.

We recommend reading this section after successfully integrating *Connext Micro* with AUTOSAR. You can optionally read this before integration to better understand the design, evaluate the approach, and assess compatibility.

At a minimum, read *Understanding the architecture* first for a high-level overview, then come here for more information.

The implementation outlined in this section provides a layered architecture for integrating *Connext Micro* into AUTOSAR Classic Platform systems. This guide describes the architectural patterns, component interactions, and design decisions.

### 14.2.1 System architecture

```
                  Application SW-C Layer
                (MCU_EXAMPLE_ASR_APP)

                        RTE Ports

                   AUTOSAR RTE Layer

                        RTE Ports
```

(continues on next page)

(continued from previous page)

```
                DDS CDD SW-C Layer
                 (MCU_DDS_CDD)

   Layer 1: AUTOSAR Template (DdsCdd.c)
   - Runnables mapped to OS tasks
   - Thin wrapper calling adapter functions


   Layer 2: Adapter (dds_cdd_adapter.c/h)
   - RTE to DDS type conversion
   - TcpIp integration
   - Initialization state machine
   - AUTOSAR system property configuration


   Layer 3: DDS Implementation (dds_impl.c/h)
   - DDS entity creation and management
   - DataWriter/DataReader operations
   - QoS configuration


                        DDS API

            RTI Connext DDS Micro Layer
  - Domain Participant
  - Topics, Publishers, Subscribers
  - DataWriters, DataReaders

                        Network I/O

            AUTOSAR TcpIp (Eth Stack)
```

### 14.2.2 RTE template layer

The RTE template layer is implemented in `MCU_DDS_CDD/dds_cdd/DdsCdd.c`. This file contains a partially-generated AUTOSAR SW-C implementation that serves as the interface between the RTE and the DDS functionality. The RTE generator creates the function signatures and header structure, while you (the integrator) provide the runnable implementations as stub templates that you must fill in.

Your primary responsibilities in this layer are to implement runnables with trigger conditions (configured in ARXML), call corresponding adapter layer functions, and provide a thin wrapper with minimal logic. The RTE generator creates runnable stubs that you implement by calling the appropriate adapter layer functions, as shown below:

```
// Initialization runnable (triggered by InitEvent trigger condition)
FUNC(void, DdsCdd_CODE) DdsCddStart(void)
```

(continues on next page)

(continued from previous page)

```
{
    DdsCdd_Adapter_Init();
}

// Timer runnable for DDS QoS maintenance (10ms periodic trigger condition)
FUNC(void, DdsCdd_CODE) DdsCddRun(void)
{
    DdsCdd_Adapter_Run();
}

// Inter-runnable trigger for processing received data
FUNC(void, DdsCdd_CODE) DdsCddRxIndication(void)
{
    Rte_IrTrigger_DdsCdd_RxIndication_ITP_DdsCdd_RxIndication();
}

// Runnable to process received data (triggered by Inter-runnable trigger)
FUNC(void, DdsCdd_CODE) DdsCddProcessData(void)
{
    NETIO_Autosar_udp_receive_callback();
}

// Data reception runnable (100ms periodic trigger condition)
// Pattern: One runnable per DDS topic you subscribe to
FUNC(void, DdsCdd_CODE) DdsCddRead_<DDS_topic>(void)
{
    DdsCdd_Adapter_Read_<DDS_topic>();
}

// Example: For PCtoMCU topic
FUNC(void, DdsCdd_CODE) DdsCddRead_PCtoMCU(void)
{
    DdsCdd_Adapter_Read_PCtoMCU();
}

// Data transmission runnable (DataReceivedEvent trigger condition)
// Pattern: One runnable per DDS topic you publish to
FUNC(void, DdsCdd_CODE) DdsCddWrite_<DDS_topic>(void)
{
    DdsCdd_Adapter_Write_<DDS_topic>();
}

// Example: For MCUtoPC topic
FUNC(void, DdsCdd_CODE) DdsCddWrite_MCUtoPC(void)
{
    DdsCdd_Adapter_Write_MCUtoPC();
}
```

### Layer 2: Adapter layer

The adapter layer implementation resides in `MCU_DDS_CDD/adaptation/dds_cdd_adapter.c` and `dds_cdd_adapter.h`. This layer acts as a bridge between AUTOSAR-specific concerns (RTE types, TcpIp, resources) and the DDS implementation.

The adapter layer handles five key responsibilities:

- It performs type conversion by mapping between RTE types and DDS types;
- It integrates with TcpIp by handling socket owner initialization and IP address management;
- It manages the initialization sequence through a state machine;
- It configures AUTOSAR-specific DDS system properties;
- It manages AUTOSAR resource allocation.

The adapter layer implements an initialization state machine that guides the system through several states:

```
TcpIpNotReady

      IP address assigned

TcpIpReady

      Set AUTOSAR system properties

SystemPropertiesSet

      Create DDS entities (disabled)

EntitiesCreated

      Enable entities

EntitiesEnabled

      Normal operation

(Running State)
```

The state machine implementation defines several states that the adapter transitions through. The state begins as `DDSCDD_STATE_TCPIP_NOT_READY` and progresses through `DDSCDD_STATE_TCPIP_READY`, `DDSCDD_STATE_SYSTEM_PROPERTIES_SET`, `DDSCDD_STATE_ENTITIES_CREATED`, and finally `DDSCDD_STATE_ENTITIES_ENABLED`:

```
typedef enum {
    DDSCDD_STATE_TCPIP_NOT_READY,
    DDSCDD_STATE_TCPIP_READY,
    DDSCDD_STATE_SYSTEM_PROPERTIES_SET,
    DDSCDD_STATE_ENTITIES_CREATED,
    DDSCDD_STATE_ENTITIES_ENABLED
```

(continues on next page)

(continued from previous page)

```
} DdsCdd_StateType;

static DdsCdd_StateType DdsCdd_State = DDSCDD_STATE_TCPIP_NOT_READY;
```

The generated runnable sequence is only part of the startup behavior. The adapter code implements explicit state guards to ensure initialization remains safe under timing variation, such as delayed IP assignment or repeated runnable execution. The adapter layer splits initialization responsibility between two functions: `DdsCdd_Adapter_Init()` and `DdsCdd_Adapter_Run()`.

`DdsCdd_Adapter_Init()` performs one-time setup phases such as setting system properties and creating DDS entities, and it transitions state only on success. `DdsCdd_Adapter_Run()` owns periodic work and enables entities when the system reaches `EntitiesCreated` state. The `DdsCdd_Adapter_Run()` function can also invoke `DdsCdd_Adapter_Init()` when state is `TcpIpReady`, providing a safe retry path from the periodic context. Any failure transitions the state to `Error` to avoid partial startup continuation.

The following code example shows how the adapter implements state guards and transitions:

```
void DdsCdd_Adapter_Init(void)
{
    if (state == DdsCdd_InitState_TcpIpNotReady ||
        state == DdsCdd_InitState_Error)
    {
        return;
    }

    if (state == DdsCdd_InitState_TcpIpReady)
    {
        if (SetSystemProperties() != 0 ||
            OSAPI_System_initialize() != RTI_TRUE)
        {
            state = DdsCdd_InitState_Error;
            return;
        }
        state = DdsCdd_InitState_SystemPropertiesSet;
    }

    if (state == DdsCdd_InitState_SystemPropertiesSet)
    {
        if (DdsImpl_CreateEntities() != 0)
        {
            state = DdsCdd_InitState_Error;
            return;
        }
        state = DdsCdd_InitState_EntitiesCreated;
    }
}

void DdsCdd_Adapter_Run(void)
{
    OSAPI_SystemAutosar_timer_callback();
```

(continues on next page)

(continued from previous page)

```
    if (state == DdsCdd_InitState_TcpIpReady)
    {
        DdsCdd_Adapter_Init();
    }
    else if (state == DdsCdd_InitState_EntitiesCreated)
    {
        if (DdsImpl_EnableEntities() == 0)
        {
            state = DdsCdd_InitState_EntitiesEnabled;
        }
        else
        {
            state = DdsCdd_InitState_Error;
        }
    }
}
```

The separation of initialization into these two functions provides several benefits. `DdsCdd_Adapter_Init()` performs one-time setup phases including system properties configuration and DDS entity creation, and it transitions state only on success. `DdsCdd_Adapter_Run()` owns the periodic work and enables entities when the system reaches the `EntitiesCreated` state. `DdsCdd_Adapter_Run()` can invoke `DdsCdd_Adapter_Init()` when state is `TcpIpReady`, providing a safe retry path from the periodic context. Any failure transitions to `Error` state to avoid partial startup continuation.

### TcpIp Configuration

This architecture expects a static TcpIp configuration where the local IP address is assigned as soon as TcpIp starts. In the documented reference approach, `Rte_Start()` is triggered from `DDS_CDD_LocalIpAddrAssignmentChg()` after IP assignment and link-up, and DDS initialization is then performed in `DdsCddStart`.

- Static IP systems (most integrations): The IP address is immediately assigned when TcpIp initializes, allowing `Rte_Start()` and `DdsCddStart` to execute quickly after network readiness is reached.
- Dynamic IP systems (DHCP): If the IP address assignment occurs later in the sequence, startup may be delayed until the callback reports assigned state. *Connext Micro* requires the local IP to be assigned prior to fully initializing.

Note the following key functions:

- `DdsCdd_Adapter_Init()`: Initialize adapter and start state machine.
- `DdsCdd_Adapter_Run()`: Advance state machine and maintain DDS QoS protocols.
- `DdsCdd_Adapter_Read_PCtoMCU()`: Poll for incoming DDS data.
- `DdsCdd_Adapter_Write_MCUtoPC()`: Publish DDS data.

- `DDS_CDD_LocalIpAddrAssignmentChg()`: TcpIp callback for IP address changes.
- `DDS_CDD_RxIndication()`: TcpIp callback for received data.

#### Layer 3: DDS implementation layer

The DDS implementation layer is contained in `MCU_DDS_CDD/dds_impl/dds_impl.c` and `dds_impl.h`. This layer handles DDS logic isolated from AUTOSAR concerns and is responsible for several key functions.

The responsibilities of this layer include entity creation to set up *DomainParticipants*, *Topics*, *Publishers*, *Subscribers*, *DataWriters*, and *DataReaders*. It also handles data operations for writing and reading DDS samples. The layer applies QoS policies from XML configuration and makes use of generated DDS type and application support code from `dds_gen/`.

The key functions implemented by this layer are as follows:

```
// Create DDS entities from XML (participant in disabled state)
int DdsImpl_CreateEntities(void);

// Enable all DDS entities (call after CreateEntities)
int DdsImpl_EnableEntities(void);

// Publish outbound data to DDS
int DdsImpl_Write_MCUtoPC(const ExampleType_t *sample);

// Read inbound data from DDS
int DdsImpl_Read_PCtoMCU(ExampleType_t *sample, int *is_valid);
```

The DDS entity hierarchy is organized as follows:

```
DomainParticipant  ("AppLibrary::MCU_DDS_CDD" - from dds_system.xml)
  Publisher
      DataWriter (MCUtoPC - outbound)
  Subscriber
       DataReader (PCtoMCU - inbound)
```

### 14.2.3 Initialization sequence

Initialization follows the detailed sequence below:

```
1. TcpIp Stack Initialization
    > IP address assigned (static configuration)

2. DDS_CDD_LocalIpAddrAssignmentChg() callback
    > Called when local IP becomes assigned

3. Rte_Start() [Called from DDS_CDD_LocalIpAddrAssignmentChg()]
    > Initialize all SW-Cs
    > Trigger InitEvent condition → DdsCddStart()
```

(continues on next page)

(continued from previous page)

```
         > Call DdsCdd_Adapter_Init()

4. First DdsCddRun() Execution (State: TCPIP_READY)
    > State machine starts/continues in adapter
    > Set AUTOSAR system properties:
        > Memory heap allocation
        > Resource management hooks
        > Timer hooks
    > State → SYSTEM_PROPERTIES_SET

5. Second DdsCddRun() Execution (State: SYSTEM_PROPERTIES_SET)
    > Call DdsImpl_CreateEntities()
        > Register APPGEN plugin (loads dds_system.xml into factory)
        > DDS_DomainParticipantFactory_create_participant_from_config()
           (creates participant + all topics, writers, readers - all disabled)
    > State → ENTITIES_CREATED

6. Third DdsCddRun() Execution (State: ENTITIES_CREATED)
    > Call DdsImpl_EnableEntities()
        > DDS_Entity_enable() on participant (cascades to all children)
    > State → ENTITIES_ENABLED

7. Normal Operation (State: ENTITIES_ENABLED)
    > DdsCddRun() maintains DDS system (10ms)
    > DdsCddRead_PCtoMCU() polls for data (100ms)
    > DdsCddWrite_MCUtoPC() publishes on trigger
```

**Note:** This document describes one valid initialization strategy. Another valid strategy is to start RTE in the normal BswM sequence and defer DDS initialization to a periodic runnable that checks network readiness.

### Ordering constraint

The startup ordering must ensure DDS initialization begins only after network readiness (IP address assigned and link is up):

1. `TcpIp_Init()`: Network stack initializes first.
2. `DDS_CDD_LocalIpAddrAssignmentChg()` callback indicates local IP assigned and link is up.
3. `Rte_Start()` is called from the callback, which triggers `DdsCddStart()`.

This ordering is necessary for several reasons. DDS requires an IP address to be available before full initialization can proceed. Starting the RTE only after network readiness helps prevent DDS startup failures. Additionally, callback-driven startup avoids race conditions between network readiness and DDS entity creation.

To implement this correctly, ensure the TcpIp local-address-assignment callback invokes `Rte_Start()` only when the target unicast IP is in assigned state (see *Appendix A: AUTOSAR*

*Integration Example* for details).

**Note:** For systems using DHCP, callback timing may be later. Keep the same network-readiness gate and avoid starting DDS initialization before assigned state.

### 14.2.4 Resource management

This section details how the DDS CDD manages memory and coordinates with AUTOSAR services to ensure thread-safe operations and predictable resource utilization.

#### Memory architecture

Memory requirements depend on number of topics, sample sizes, history depth, discovery requirements, and resource limits. See *Customizing for your application* and *Memory sizing* for detailed sizing guidelines.

#### AUTOSAR resource synchronization

The DDS CDD uses a single OS resource for internal synchronization (mutex protection). All DDS operations that access shared data structures are guarded by this one resource.

```
/* Actual implementation: single resource for mutex synchronization */
system_property.psl_property.sync_type = OSAPI_AUTOSAR_SYNCKIND_RESOURCES;
system_property.psl_property.mutex_resource_id = OsResource_000;
```

1 STANDARD Resource (typically named `OsResource_000`) must be configured in the OS. All DDS entity creation and data read/write operations are protected by this single resource. See *OS configuration* for configuration details.

When any DDS API is called (participant creation, entity operations, data write/read), the system automatically acquires `OsResource_000`, performs the operation, and releases the resource. This ensures thread-safe access from multiple AUTOSAR tasks and the background DDS housekeeping task.

#### TcpIp socket management

The socket owner is configured with the following macro definition:

```
#define DDSCDD_SOCKET_OWNER_NAME "DdsSocketOwner"
```

The DDS CDD requires a total of 4 UDP sockets when operating in unicast mode, or 5 UDP sockets when operating in multicast mode. These sockets use dynamic port allocation and must register Rx indication callbacks to handle incoming data.

### 14.2.5 Timing architecture

This section covers runnable scheduling, period requirements, task priority guidelines, and timing considerations to ensure proper DDS protocol maintenance and predictable end-to-end latency within the AUTOSAR environment.

#### Runnable timing configuration

```
DdsCddStart
 > Trigger Condition: InitEvent
 > Execution: Once at startup

DdsCddRun
 > Trigger Condition: TimingEvent (10ms periodic)
 > Purpose: DDS QoS maintenance
 > Tasks: Discovery, liveliness, reliability protocol

DdsCddRead_PCtoMCU
 > Trigger Condition: TimingEvent (100ms periodic)
 > Purpose: Poll for incoming data
 > Tasks: Check DataReader, write to RTE

DdsCddWrite_MCUtoPC
 > Trigger Condition: DataReceivedEvent on input port
 > Purpose: Publish data to network
 > Tasks: Read from RTE, publish via DataWriter
```

**Note:** These trigger conditions are configured in DdsCdd ARXML during RTE configuration (see *Configuring RTE*). The RTE generator automatically creates the corresponding OS alarms and events from these trigger condition specifications.

Note the following timing considerations:

**Warning:** If DDS initialization is executed inside a periodic runnable, the first activation can exceed the nominal runnable period (for example, a 10 ms runnable can run longer during initialization). Include this in watchdog, deadline monitoring, and task load budget analysis.

1. 10ms DdsCddRun Period:
   - Must be fast enough for DDS protocol maintenance.
   - Affects discovery time and reliability protocol.
   - Too slow: Increased latency, potential data loss.
   - Too fast: Unnecessary CPU overhead.
2. 100ms DdsCddRead Period:

- Determines maximum data reception rate.
- Should match application requirements.
- Consider jitter and worst-case execution time.

3. DataReceivedEvent Trigger Condition:
   - Runnable triggered immediately when app writes data to RTE port.
   - Minimizes end-to-end latency.
   - No polling overhead.

DDS CDD runnables should have higher priority than application SW-Cs, but lower priority than critical BSW tasks (Ethernet driver, TcpIp stack). This ensures timely network communication without blocking critical system functions.

### 14.2.6 Error handling strategy

During initialization, if `DdsImpl_Init()` encounters errors, the system logs the error, remains in the current state, and retries on the next `DdsCddRun()` cycle. This strategy avoids catastrophic failures during startup:

```
if (DdsImpl_Init() != DDS_RETCODE_OK) {
    // Log error
    // Remain in current state
    // Retry on next DdsCddRun() cycle
}
```

At runtime, the system handles write operations by checking the return code from `DdsImpl_Write_MCUtoPC()`. If the write operation fails with a return code other than `DDS_RETCODE_OK`, the data is lost and the application must implement retry logic if data transmission is critical. The error is logged for diagnostics to help identify patterns or persistent issues:

```
DDS_ReturnCode_t result = DdsImpl_Write_MCUtoPC(&data);
if (result != DDS_RETCODE_OK) {
    // Data lost, application should implement retry logic
    // Log error for diagnostics
}
```

Read operations similarly check the return code from `DdsImpl_Read_PCtoMCU()`. If the result is `DDS_RETCODE_NO_DATA`, this is a normal condition indicating that no new data is currently available. However, if the result indicates any other error, the system logs it for diagnostics to aid in troubleshooting. In all cases, errors propagate through return codes, allowing the application to decide its own retry strategy based on system requirements:

```
DDS_ReturnCode_t result = DdsImpl_Read_PCtoMCU(&data);
if (result == DDS_RETCODE_NO_DATA) {
    // No new data available (normal condition)
} else if (result != DDS_RETCODE_OK) {
    // Error reading data, log for diagnostics
}
```

### 14.2.7 Scalability considerations

Keep the following in mind when scaling your system architecture.

When adding topics to your system, update `dds_system.xml` with new topic definitions, then regenerate DDS type and application support code (both generated from DDS-XML). Add new port interfaces to ARXML, create new adapter functions in Layer 2 and Layer 3, add new runnables to Layer 1, and increase memory allocation if needed.

For a multi-topic architecture, ensure the system is organized across the three layers, as shown below:

```
// Layer 3: Support multiple topics
DDS_ReturnCode_t DdsImpl_Write_Topic1(const Type1* data);
DDS_ReturnCode_t DdsImpl_Write_Topic2(const Type2* data);
DDS_ReturnCode_t DdsImpl_Read_Topic1(Type1* data);
DDS_ReturnCode_t DdsImpl_Read_Topic2(Type2* data);

// Layer 2: Multiple adapter functions
void DdsCdd_Adapter_Write_Topic1(void);
void DdsCdd_Adapter_Write_Topic2(void);
void DdsCdd_Adapter_Read_Topic1(void);
void DdsCdd_Adapter_Read_Topic2(void);

// Layer 1: Multiple runnables
FUNC(void, DdsCdd_CODE) DdsCddWrite_Topic1(void);
FUNC(void, DdsCdd_CODE) DdsCddWrite_Topic2(void);
FUNC(void, DdsCdd_CODE) DdsCddRead_Topic1(void);
FUNC(void, DdsCdd_CODE) DdsCddRead_Topic2(void);
```

## 14.3 Appendix C: Package Contents

This section provides an overview of the installation package contents, as well as the resultant directory structure in the *Connext Micro* installation.

### 14.3.1 Host bundle

When installed, the host bundle (`rti_connext_dds_micro-<version>-host.rtipkg`) adds the following to the *Connext Micro* directory:

```
--rti_connext_dds-<version>/
  |
  +--rti_connext_dds_micro-<version>/
      |--bin/
      |--doc/
      |--example/
      |--include/
      |--resource/
      |--rtiddsgen/
```

(continues on next page)

(continued from previous page)

```
     |--rtiddsmag/
     |--CMakeLists.txt
     |--ReadMe.html
     +--src/
        +--rti_me_psl
```

- The `bin/` directory contains scripts for *rtiddsgen*, *rtiddsmag*, and rtime-make.
- The `doc/` directory contains this documentation, as well as the C and C++ API References.
- The `example/` directory contains buildable example applications, as well as instructions on how to build and run them. Refer to *Provided examples* for more information.
- The `include/` directory contains the public header files to compile applications.
- The `resource/` directory contains the build system used for the examples and Platform Support Libraries (PSL).
- The `rtiddsgen/` directory contains an IDL compiler for type support code.
- The `rtiddsmag/` directory contains a tool for generating application code from XML descriptions.
- CMakeLists.txt is the main input file to CMake and is used to generate build files.
- `ReadMe.html` opens this documentation.
- `src/` contains the source files for all supported Platform Support Libraries (PSL); refer to *Target bundle* for more information on the PSL. These can be recompliled for specific platform configurations, as described in *Building the PSL*.

### 14.3.2 Target bundle

When installed, the target bundle (`rti_connext_dds_micro-<version>-target-<architecture>.rtipkg`) adds the following to the *Connext Micro* directory:

```
--rti_connext_dds-<version>/
    |--rti_connext_dds_micro-<version>/
            |--lib/
            |    |--<arch>/
            |    |     +---<arch libraries>
            |    |--<arch>CERT/
            |    |     +---<arch CERT libraries>
            |    |--<arch>-<PSL>/
            |    |     +-- <arch PSL libraries>
            |    |--<arch>CERT-<PSL>/
            |    |     +-- <arch CERT PSL libraries>
            |
            |--sbom/
```

- The `lib/` directory contains the libraries needed to build *Connext Micro*.

  - `<arch>` contains pre-built Static and Dynamic (where supported) Release and Debug libraries. These may be integrated libraries or Platform Independent Libraries; see *Library descriptions* below.
  - `<arch>CERT` contains pre-built CERT profile Release and Debug libraries, if they exist for the specific architecture. These may be integrated libraries or Platform Independent Libraries; see *Library descriptions* below.
  - `<arch>-<PSL>` contains pre-built Platform Support Libraries for the specific architecture.
  - `<arch>CERT-<PSL>` contains CERT profile Platform Support Libraries, if they exist for the specific architecture.
- The `sbom/` directory contains the Software Bill of Materials (SBOM) for the target libraries.

### 14.3.3 Lightweight Security Plugin host bundle

When installed, the Lightweight Security Plugin host bundle (`rti_connext_dds_micro-<version>-lw-security-host.rtipkg`) adds the following to the *Connext Micro* directory:

```
--rti_connext_dds-<version>/
    |--rti_connext_dds_micro-<version>/
            |--include/
            |      |--rti_me/
            |           |--dds_psk/
            |           |--rti_me_psl/
            |                |--pskpsl/
            |--rtiddsgen/
            |      +--<security-specific example templates>
```

- The `include/` directory contains security-related header files to use the security binaries. These header files do not include the security interfaces.
- The `rtiddsgen/` directory includes security-specific example templates.

### 14.3.4 Lightweight Security Plugin target bundle

When installed, the Lightweight Security Plugin target bundle (`rti_connext_dds_micro-<version>-lw-security-target-openssl-3.5-<architecture>.rtipkg`) adds the following to the *Connext Micro* directory:

```
--rti_connext_dds-<version>/
    |--rti_connext_dds_micro-<version>/
            |--lib/
            |      |--<arch>/
            |      |     +--<arch PSK Security PIL>
            |      |--<arch>-<PSL>/
            |      |     +--<arch Transform PSL libraries>
            |      |--<arch>CERT/
```

(continues on next page)

(continued from previous page)

```
            |      |     +--<arch CERT PSK Security PIL>
            |      |--<arch>CERT-<PSL>/
            |      |     +--<arch CERT Transform PSL libraries>
            |
            |--sbom/
                   +--<*ddspsk*>
                   +--<*pskpsl*>
```

- The `lib/` directory includes the libraries for your architecture and platform, including the pre-built Transform PSL (`pskpsl_openssl3`).
- The `sbom/` directory includes the Software Bill of Materials (SBOM) for the security target libraries.

## 14.4 Appendix D: Directory Structure

The complete, default *Connext Micro* installation is structured as shown below:

```
+--rti_connext_dds-<version>/
   |
   +--rti_connext_dds_micro-<version> (rti_connext_dds_micro-<version>-host.rtipkg)
      |--bin/
      |--doc/
      |--example/
      |--include/
      |--resource/
      |--rtiddsgen/
      |--rtiddsmag/
      |--CMakeLists.txt
      |--ReadMe.html
      |--sbom/
      +--src/
      |  +-- rti_me_psl
      +--lib (rti_connext_dds_micro-<version>-target-<arch>.rtipkg)
         +--<arch>
         |  +---<arch libraries>
         +--<arch>CERT
         |   +---<arch libraries>
         +--<arch>-<PSL>
         |   +-- <arch PSL libraries>
         +--<arch>CERT-<PSL>
               +-- <arch PSL libraries>
```

This directory structure is recommended and *should* be used because:

- The source bundle includes a helper script to run CMake that expects this directory structure.
- This directory structure supports multiple architectures.