RTI Connext Micro

User's Manual

Version 4.1.0

Contents

RTI® Connext® DDS Micro provides a small-footprint, modular messaging solution for resource-limited devices that have limited memory and CPU power, and may not even be running an operating system. It provides the communications services that developers need to distribute time-critical data. Additionally, *Connext Micro* is designed as a certifiable component in high-assurance systems.

Key benefits of *Connext Micro* include:

- Accommodations for resource-constrained environments.
- Modular and user extensible architecture.
- Designed to be a certifiable component for safety-critical systems.
- Seamless interoperability with *RTI Connext Professional*.

Chapter 1

Contents

1.1 Introduction

1.1.1 What is RTI Connext Micro?

RTI Connext Micro is network middleware for distributed real-time applications. It provides the communications service programmers need to distribute time-critical data between embedded and/or enterprise devices or nodes. *Connext Micro* uses the publish-subscribe communications model to make data distribution efficient and robust. *Connext Micro* simplifies application development, deployment and maintenance and provides fast, predictable distribution of time-critical data over a variety of transport networks. With *Connext Micro*, you can:

- Perform complex one-to-many and many-to-many network communications.
- Customize application operation to meet various real-time, reliability, and quality-of-service goals.
- Provide application-transparent fault tolerance and application robustness.
- Use a variety of transports.

Connext Micro implements the Data-Centric Publish-Subscribe (DCPS) API within the OMG's Data Distribution Service (DDS) for Real-Time Systems. DDS is the first standard developed for the needs of real-time systems. DCPS provides an efficient way to transfer data in a distributed system.

With *Connext Micro*, systems designers and programmers start with a fault-tolerant and flexible communications infrastructure that will work over a wide variety of computer hardware, operating systems, languages, and networking transport protocols. *Connext Micro* is highly configurable so programmers can adapt it to meet the application's specific communication requirements.

Publish-Subscribe Middleware

Connext Micro is based on a publish-subscribe communications model. Publish-subscribe (PS) middleware provides a simple and intuitive way to distribute data. It decouples the software that creates and sends data—the data publishers—from the software that receives and uses the data—the data subscribers. Publishers simply declare their intent to send and then publish the data. Subscribers declare their intent to receive, then the data is automatically delivered by the middleware. Despite the simplicity of the model, PS middleware can handle complex patterns of information flow. The use of PS middleware results in simpler, more modular distributed applications. Perhaps most importantly, PS middleware can automatically handle all network chores, including connections, failures, and network changes, eliminating the need for user applications to program of all those special <cases. What experienced network middleware developers know is that handling special cases accounts for over 80% of the effort and code.

1.1.2 Supported DDS Features

Connext Micro supports a subset of the DDS DCPS standard. A brief overview of the supported features are listed here. For a detailed list, please refer to the C API Reference and C++ API Reference.

DDS Entity Support

Connext Micro supports the following DDS entities. Please refer to the documentation for details.

- DomainParticipantFactory
- DomainParticipant
- Topic
- Publisher
- Subscriber
- DataWriter
- DataReader

DDS QoS Policy Support

Connext Micro supports the following DDS Qos Policies. Please refer to the documentation for details.

- DDS_DataReaderProtocolQosPolicy
- DDS_DataReaderResourceLimitsQosPolicy
- DDS_DataWriterProtocolQosPolicy
- DDS_DataWriterResourceLimitsQosPolicy
- DDS_DeadlineQosPolicy

- DDS_DiscoveryQosPolicy
- DDS_DomainParticipantResourceLimitsQosPolicy
- DDS_DurabilityQosPolicy
- DDS_DestinationOrderQosPolicy
- DDS_EntityFactoryQosPolicy
- DDS_HistoryQosPolicy
- DDS_LivelinessQosPolicy
- DDS_OwnershipQosPolicy
- DDS_OwnershipStrengthQosPolicy
- DDS_ReliabilityQosPolicy
- DDS_ResourceLimitsQosPolicy
- DDS_RtpsReliableWriterProtocol_t
- DDS_SystemResourceLimitsQosPolicy
- DDS_TopicDataQosPolicy
- DDS_TransportQosPolicy
- DDS_UserDataQosPolicy
- DDS_UserTrafficQosPolicy
- DDS_WireProtocolQosPolicy

1.1.3 RTI Connext DDS Documentation

Throughout this document, we may suggest reading sections in other *RTI Connext* documents. These documents are in your *RTI Connext* installation directory under **rti-connext-dds-<version>/doc/manuals**. A quick way to find them is from *RTI Launcher's* Help panel, select "Browse Connext Documentation".

Since installation directories vary per user, links are not provided to these documents on your local machine. However, we do provide links to documents on the RTI Documentation site for users with Internet access.

New users can start by reading Parts 1 (Introduction) and 2 (Core Concepts) in the RTI Connext Core Libraries User's Manual. These sections teach basic DDS concepts applicable to all RTI middleware, including *RTI Connext Professional* and *RTI Connext Micro*. You can open the RTI Connext Core Libraries User's Manual from *RTI Launcher's* Help panel.

The RTI Community provides many resources for users of DDS and the RTI Connext family of products.

1.1.4 OMG DDS Specification

For the original DDS reference, the OMG DDS specification can be found in the OMG Specifications under “Data Distribution Service”.

1.1.5 Other Products

RTI Connext Micro is one of several products in the *RTI Connext* family of products:

RTI Connext Cert is a subset of *RTI Connext Micro*. *Connext Cert* does not include the following features because Certification Evidence is not yet available for them. If you require Certification Evidence for any of these features, please contact RTI.

- C++ language API.
- Multi-platform support.
- Dynamic endpoint discovery.
- delete() APIs (e.g. delete_datareader())

RTI Connext Professional addresses the sophisticated databus requirements in complex systems including an API compliant with the Object Management Group (OMG) Data Distribution Service (DDS) specification. DDS is the leading data-centric publish/subscribe (DCPS) messaging standard for integrating distributed real-time applications. *Connext Professional* is the dominant industry implementation with benefits including:

- OMG-compliant DDS API
- Advanced features to address complex systems
- Advanced Quality of Service (QoS) support
- Comprehensive platform and network transport support
- Seamless interoperability with *Connext Micro*

RTI Connext Professional includes rich integration capabilities:

- Data transformation
- Integration support for standards including JMS, SQL databases, file, socket, Excel, OPC, STANAG, LabVIEW, Web Services and more
- Ability for users to create custom integration adapters
- Optional integration with Oracle, MySQL and other relational databases
- Tools for visualizing, debugging and managing all systems in real-time

RTI Connext Professional also includes a rich set of tools to accelerate debugging and testing while easing management of deployed systems. These components include:

- Administration Console
- Distributed Logger

- Monitor
- Monitoring Library
- Recording Service

1.2 Installation

1.2.1 Installing the RTI Connext Micro Packages

Note: *Connext Micro* 4.1.0 is distributed differently from earlier releases. Specifically, only the platform source code is included instead of the full source. Refer to the *Overview of the Target Bundle* for more details.

RTI 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`

You must first install *RTI Connext Professional* 7.3.0 before installing *Connext Micro* 4.1.0 packages. To install the *Connext Micro* packages:

1. Open the *RTI Launcher* for *Connext Professional* 7.3.0.
2. Navigate to the **Configuration** tab.
3. Select **Install RTI Packages**.
4. In the popup window, click on the + icon and add both `.rtipkg` files to the installation queue.
5. Select **Install**.

Once installed, you will find a directory called `rti_connext_dds_micro-<version>` in the *Connext Professional* installation directory.

Note: A Java Runtime Environment (JRE) is needed to run the IDL compiler *rtiddsgen* and the Micro Application Generator *rtiddsmag*. By default, *Connext Micro* will use the JRE that is already included in the *Connext Drive* or *Connext Professional* installation where *Connext Micro* is installed.

If you prefer a different JRE, you must set the environment variable `JREHOME` to the path of the specified JRE.

Warning: If you plan to recompile the platform source code, RTI strongly recommends that you copy the *Connext Micro* installation directory outside of the *Connext Professional* installation. It may not be desirable to build *Connext Micro* libraries inside the *Connext Professional* directory due to patches, lack of write access, or other factors.

1.2.2 Overview of the Host Bundle

This section provides an overview of the host package contents, as well as the resultant directory structure in the *Connext Micro* installation.

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>/
      |--doc/
      |--example/
      |--include/
      |--resource/
      |--rtiddsgen/
      |--rtiddsmag/
      |--CMakeLists.txt
      |--ReadMe.html
      +--src/
         +--rti_me_psl
```

- 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.
- 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 *Overview of the Target Bundle* for more information on the PSL. These can be recompliled for specific platform configurations, as described in *Building the PSL*.

1.2.3 Overview of the Target Bundle

This section provides an overview of the target package contents, the types of libraries included in the bundle, and the library names and descriptions.

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 libraries>
          +--<arch>-<PSL>
          |   +-- <arch PSL libraries>
          +--<arch>CERT-<PSL>
                +-- <arch PSL libraries>
```

- 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 types* 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 types* 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.

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:

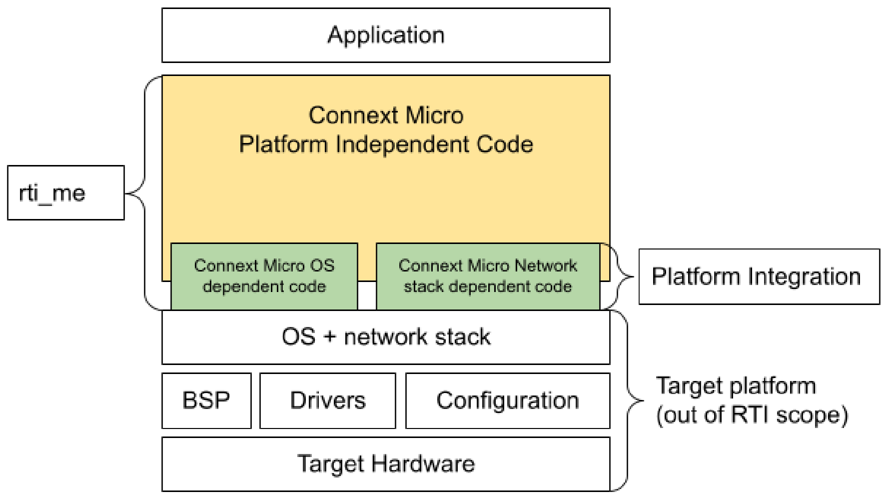


Figure 1.1: An overview of rti_me as an integrated library

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:

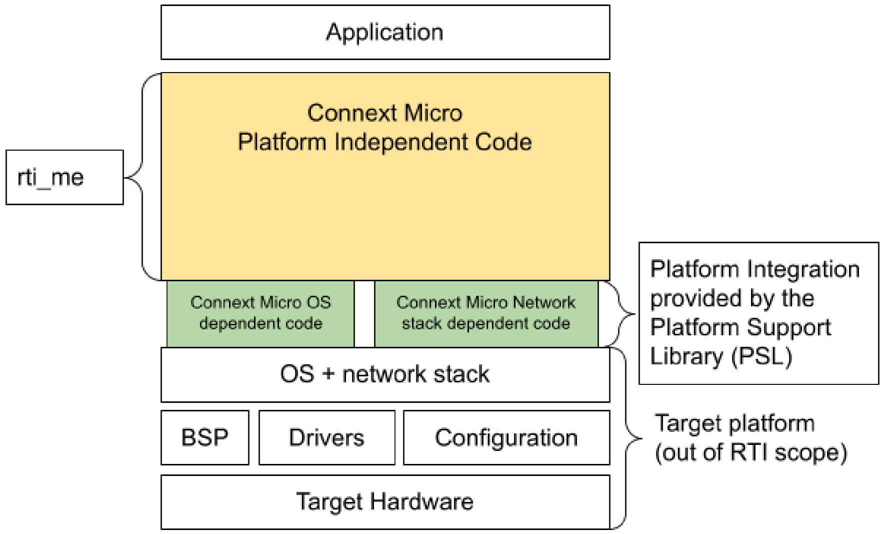


Figure 1.2: An overview of split libraries

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.

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 1.1: 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)

1.2.4 Overview of the Source Bundle

> **Warning:** This section only applies to the *Connext Micro* source bundle (`rti_connext_dds_micro-<version>-source.zip`). For other bundles, refer to *Overview of the Host Bundle* and *Overview of the Target Bundle*.

When unzipped, the source bundle (`rti_connext_dds_micro-<version>-source.zip`) creates the following directory structure:

```
--rti_connext_dds_micro-<version>/
    |--doc
    |--include/
    |--resource/
    |--rtiddsgen/
    |--rtiddsmag/
    |--src/
    |--CMakeLists.txt
    |--ReadMe.html
```

- The `doc/` directory contains this documentation, as well as the C and C++ API References.
- 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.

- The `src/` directory contains the source files for all supported Platform Independent Libraries and Platform Support Libraries.
- CMakeLists.txt is the main input file to CMake and is used to generate build files.
- `ReadMe.html` opens this documentation.

1.2.5 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)
      |--doc/
      |--example/
      |--include/
      |--resource/
      |--rtiddsgen/
      |--rtiddsmag/
      |--CMakeLists.txt
      |--ReadMe.html
      +--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[1] because:

- The source bundle includes a helper script to run CMake that expects this directory structure.
- This directory structure supports multiple architectures.

1.3 Getting Started

Once *Connext Micro* is installed, choose any one of the following paths to proceed:

- Follow the examples in this section, using the precompiled libraries in your *Connext Micro* installation. RTI recommends this route to familiarize yourself with *Connext Micro*.
- If you want to build your own Platform Support Library (PSL), refer to *Building the PSL*, then return to this section and follow the examples.

[1] This applies to builds using CMake.

- If you have a source bundle, refer to *Building the source*, then return to this section and follow the examples.

This section will help you get started with *Connext Micro* by using the example applications that are included in your *Connext Micro* installation. These are HelloWorld examples that can be altered to suit your preferences.

Once you have used the examples in this chapter to familiarize yourself with *Connext Micro*'s features, you are ready to develop your own applications as described in *Developing Applications*.

1.3.1 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_mag_dpse**: A HelloWorld example that uses MAG to generate a dpse application.
- **HelloWorld_mag_dpde**: A HelloWorld example that uses MAG to generate a dpde application.
- **HelloWorld_mag_static_udp**: A HelloWorld example that uses MAG to generate a static udp application.
- **HelloWorld_mag_shared_memory**: A HelloWorld example that uses MAG to generate a shared memory application.

Consult the `README.txt` file included with each example for instructions on how to build and run the application.

In addition to the provided examples, the RTI *Code Generator* available with *Connext Micro* can generate example DDS applications with a type definition file as input. For more information, see *Generating Examples*.

1.3.2 Generating Examples

The RTI *Code Generator* (also referred to as *rtiddsgen*) included with *Connext Micro* can generate DDS example applications with a type definition file as input.

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
```

As an example, the following command will generate an example in the C language, using the `waitsets` custom template instead of the default template:

```
List of example templates per language:
    - C:
        - cert
        - dpse
        - shared_memory
        - static_udp
        - waitsets
        - crc
        - zcv2
    - C++:
        - dpse
        - waitsets
    - C++ Namespace:
```

(continues on next page)

(continued from previous page)

```
        - dpse
        - waitsets
```

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 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-endpoint discovery (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**

  Identical to the default template, except that the discovery of endpoints is done with static-endpoint discovery (DPSE). Static-endpoint discovery uses function calls to statically assert information about remote endpoints belonging to remote *DomainParticipants*.

- **secure**

  Identical to the default template, except that this example uses secure communication.

- **shared_memory**

  Identical to the default template, except that the only transport used is shared memory. Because the UDP transport is disabled and only the shared memory transport is enabled, both the publisher and subscriber applications need to run in the same OS.

- **static_udp**

  Identical to the default template, except that this example uses a static UDP 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…).

- **waitsets** The only difference from the default template is that 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** Identical to the default template, except that the configuration of the CRC settings. The CRC QoS enables the generation of a checksum value based on the data being transmitted. It also determines which checksums are allowed and whether they should be sent.
- **zcv2** Identical to the default template, with the added Zero Copy v2 interface. This example performs discovery over UDP. User traffic is handled using the Zero Copy v2 interface.

How to compile the generated examples

Before compiling, set the environment variable `RTIMEHOME` to the *Connext Micro* installation directory.

Depending on the number of network interfaces installed on the local machine, you might need to restrict which interfaces are used by *Connext Micro*. Use the option `-udp_intf <interface name>` when running the example.

The *Connext Micro* source bundle includes rtime-make (on Linux® and macOS® systems) or rtime-make.bat (on Windows® systems) to simplify invocation of CMake. This script is a convenient way to invoke CMake with the correct options. For example:

Linux

```
cd <directory with generated example>

rtime-make --config <Debug|Release> --build --target armv8leElfgcc7.3.0-Linux4 --source-
↪dir . \
     -G "Unix Makefiles" --delete [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] \
     [-DRTIME_LINK_SHMEM_LIBS=true]
```

macOS

```
cd <directory with generated example>

rtime-make --config <Debug|Release> --build --target x86_64leMachOclang15.0-Darwin23 --
↪source-dir . \
     -G "Unix Makefiles" --delete [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] \
     [-DRTIME_LINK_SHMEM_LIBS=true]
```

Windows

```
cd <directory with generated example>

rtime-make.bat --config <Debug|Release> --build --target x86_64lePEvs2017-Win10 --source-
↪dir . \
     -G "Visual Studio 15 2017" --delete [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE_eq_
↪true] \
     [-DRTIME_LINK_SHMEM_LIBS_eq_true]
```

> **Warning:** RTI recommends using the toolchain file that matches the target architecture to compile the generated examples.

For example, if the target architecture is `--target armv8leElfgcc7.3.0-Linux4`, then the example applications should be compiled with the `armv8leElfgcc7.3.0-Linux4` toolchain file. Failing to do so may cause warnings.

The executable can be found in the directory 'objs'.

It is also possible to compile using CMake, e.g., when the *Connext Micro* source bundle is not installed.

Linux

```
cmake [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] [-DRTIME_LINK_SHMEM_LIBS=true] \
      [-DCMAKE_BUILD_TYPE=<Debug|Release>] -G "Unix Makefiles" -B./<your build directory>
↪ -H. -DRTIME_TARGET_NAME=armv8leElfgcc7.3.0-Linux4

cmake --build ./<your build directory> [--config <Debug|Release>]
```

macOS

```
cmake [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] [-DRTIME_LINK_SHMEM_LIBS=true] \
      [-DCMAKE_BUILD_TYPE=<Debug|Release>] -G "Unix Makefiles" -B./<your build directory>
↪ -H. -DRTIME_TARGET_NAME=x86_64leMachOclang15.0-Darwin23

cmake --build ./<your build directory> [--config <Debug|Release>]
```

Windows

```
cmake [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] [-DRTIME_LINK_SHMEM_LIBS=true] \
      [-DCMAKE_BUILD_TYPE=<Debug|Release>] -G "Visual Studio 15 2017" -B./<your build␣
↪directory> -H. -DRTIME_TARGET_NAME=x86_64lePEvs2017-Win10

cmake --build .\<your build directory> [--config <Debug|Release>]
```

The executable can be found in the directory 'objs'.

The following options are accepted:

- `-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true` adds a rule to regenerate type support plugin source files if the input file with the type definition changes. Default value is 'false'.
- `-DRTIME_LINK_SHMEM_LIBS=true` adds a dependency to the shared memory transport libraries. This option shall be used only with the shared memory example. The default value is 'false'.

How to run the generated examples

By default, the example uses all available interfaces to receive samples. This can cause communication problems if the number of available interfaces is greater than the maximum number of interfaces supported by *Connext Micro*. For this reason, it is recommended to restrict the number of interfaces used by the application. Use the option `-udp_intf <interface name>` when running the example.

For example, if the example has been compiled for Linux i86Linux2.6gcc4.4.5, run the subscriber with this command:

```
objs/armv8leElfgcc7.3.0-Linux4/<Type definition file name>_subscriber [-domain
↪<Domain_ID>] [-peer <address>] \
             [-sleep <sleep_time>] [-count <seconds_to_run>] [-udp_intf
↪<interface name>]
```

and run the publisher with this command:

```
objs/armv8leElfgcc7.3.0-Linux4/<Type definition file name>_publisher [-domain
↪<Domain_ID> -peer <address>] \
             [-sleep <sleep_time>] [-count <seconds_to_run>] [-udp_intf
↪<interface name>]
```

Note: Shared memory examples only accept the following options:

- `[-domain <Domain_ID>]`
- `[-sleep <sleep_time>]`
- `[-count <seconds_to_run>]`

1.4 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 *User's Manual*.

1.4.1 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.

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.

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
```

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_rhsm, rti_me_whsm, and rti_me (always required)
- RTIMEHOME/lib/<arch>-<PSL>/
  7. rti_me_netiopsl (when building with a Platform Independent Library)
  8. 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_cpp, rti_me_netiosdm, rti_me_netiozcopy, rti_me_netioshmem, rti_me_discdpde, rti_me_discdpse, rti_me_rhsm, rti_me_whsm, and rti_me (always required)
- RTIMEHOME/lib/<arch>-<PSL>/
  3. rti_me_netiopsl_cpp (when building with a Platform Independent Library)
  4. rti_me_netiopsl (when building with a Platform Independent Library)
  5. rti_me_ospsl (when building with a Platform Independent Library)

1.4.2 Define a Data Type

To distribute data using *Connext Micro*, you must first define a data type, then run the *rtiddsgen* utility. This utility will generate the type-specific support code that *Connext Micro* needs and the code that makes calls 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 chars:

```
struct HelloWorld {
    long id; //@key
    string<128> msg;
    sequence<octet, 1000> image;
};
```

For more details, see *Data Types* in the *User's Manual*.

1.4.3 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-7.3.0/rti_connext_micro-4.0.1/rtiddsgen/scripts*.

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.
- The `-language` option specifies the language of the generated code. *Connext Micro* supports C and 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**. Support operations to generate a type-specific *DataWriter* and *DataReader*, and to register the type with a DDS *DomainParticipant*.

This release of *Connext Micro* supports a new way to generate support code for IDL Types that will generate a TypeCode object containing information used by an interpreter to serialize and deserialize samples. Prior to this release, the code for serialization and deserialization was generated for each type. To disable generating code to be used by the interpreter, use the `-interpreted 0` command-line option to generate code. This option generates code in the same way as was done in previous releases.

For more details, see *Generating Type Support with rtiddsgen* in the *User's Manual*.

1.4.4 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*).

Registry Configuration

The DomainParticipantFactory, in addition to its standard role of creating and deleting *Domain-Participants*, contains the RT Registry that a new application registers with some necessary components.

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.

Connext Micro automatically registers components that provide necessary functionality. These include components for DDS *Writers* and *Readers*, the RTPS protocol, and the UDP transport.

In addition, every DDS application must register three components:

- **Writer History**. Queue of written samples of a *DataWriter*. Must be registered with the name "wh".
- **Reader History**. Queue of received samples of a *DataReader*. Must be registered with the name "rh".
- **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, "wh",
                          WHSM_HistoryFactory_get_interface(), NULL, NULL))
{
    /* failure */
}

/* Register Reader History */
if (!RT_Registry_register(registry, "rh",
                          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  */
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  */
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 the *Application Generation Using XML* section in the User's Manual.

1.4.5 Configure UDP Transport

You may need to configure the UDP transport component that is pre-registered by *RTI Connext Micro*. To change the properties of the UDP transport, first the UDP component has be unregistered, then the properties have to be updated, and finally the component must be re-registered with the updated properties.

Example code:

- Unregister the pre-registered UDP component:

```
/* Unregister the pre-registered UDP component */
if (!RT_Registry_unregister(registry, "_udp", NULL, NULL))
{
    /* failure */
}
```

- Configure 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 send/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");
}
else
{
    /* failure */
}
```

- Re-register UDP component with updated properties:

```
if (!RT_Registry_register(registry, "_udp",
                     UDP_InterfaceFactory_get_interface(),
                    (struct RT_ComponentFactoryProperty*)udp_property, NULL))
{
    /* failure */
}
```

For more details, see the *Transports* section in the User's Manual.

1.4.6 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.
- **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 */
if (!RT_ComponentFactoryId_set_name(&dp_qos.discovery.discovery.name, "dpde"))
{
     /* 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("239.255.0.1");

/* Resource 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;

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

(continues on next page)

(continued from previous page)

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

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, preferably the same as its IDL defined name.

```
DDS_ReturnCode_t retcode;

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

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)
{
```

(continues on next page)

(continued from previous page)

```
    /* failure */
}
```

DPSE Discovery: Assert Remote Participant

DPSE Discovery relies on the application to specify the other, or 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.

```
/* Enable 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 the *DDS Domains* section in the User's Manual.

1.4.7 Create Publisher

A publishing application needs to create a DDS *Publisher* and then a *DataWriter* for each *Topic* it wants to publish.

In *Connext Micro*, PublisherQos in general contains no policies that need to be customized, while DataWriterQos does contain several customizable policies.

- 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 the *Sending Data* section in the User's Manual.

1.4.8 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 */
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,
                                    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");
    }
}
```

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");

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 */
}
```

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!");
```

(continues on next page)

(continued from previous page)

```
/* 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 the *Sending Data* section in the User's Manual.

1.4.9 Create Subscriber

A subscribing application needs to create a DDS *Subscriber* and then a *DataReader* for each *Topic* to which it wants to subscribe.

In *Connext Micro*, SubscriberQos in general contains no policies that need to be customized, while DataReaderQos does contain several customizable policies.

```
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 the *Receiving Data* section in the User's Manual.

1.4.10 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 */
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;
```

(continues on next page)

(continued from previous page)

```
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);
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 DDS_SampleInfo);
    struct HelloWorldSeq sample_seq =
        DDS_SEQUENCE_INITIALIZER(HelloWorld);
```

(continues on next page)

(continued from previous page)

```
    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);

        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);
}
```

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;
```

(continues on next page)

(continued from previous page)

```
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.

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. (Note: the code snippet below is taken from the shipped HelloWorld_dpde_waitset example).

```
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(struct DDS_ConditionSeq);

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

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

(continues on next page)

(continued from previous page)

```
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);

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));
             }
```

(continues on next page)

(continued from previous page)

```
            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;
    }
}
```

Filtering Samples

In lieu of supporting Content-Filtered Topics, a DataReaderListener in *Connext Micro* provides callbacks to do application-level filtering per sample.

- **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,
```

(continues on next page)

(continued from previous page)

```
    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 the *Receiving Data* section in the User's Manual.

1.5 User's Manual

1.5.1 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/.

   Define a type programmatically at run time.

   This method may be appropriate for applications with dynamic data description needs: applications for which types change frequently or cannot be known ahead of time.

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 the section on Working with DDS Data Samples in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

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. They can be composed of any of the primitive data types listed below or of other user data types.

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.

- 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

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 HTML documentation.

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.

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*, available here if you have Internet access).

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*, available here if you have Internet access).

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 here for more information.

Extensible Types (X-Types) 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 X-Types 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 X-Types 1.2 features cannot be supported, such as:
  - Type equivalence
  - String-length matching and truncation
  - Sequence-length matching and truncation

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 (available here if you have Internet access).

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

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 (available here if you have Internet access).

1.5.2 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 (QosPolicies). 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 4.1: Overview of DDS Entities* presents an overview in a UML diagram.

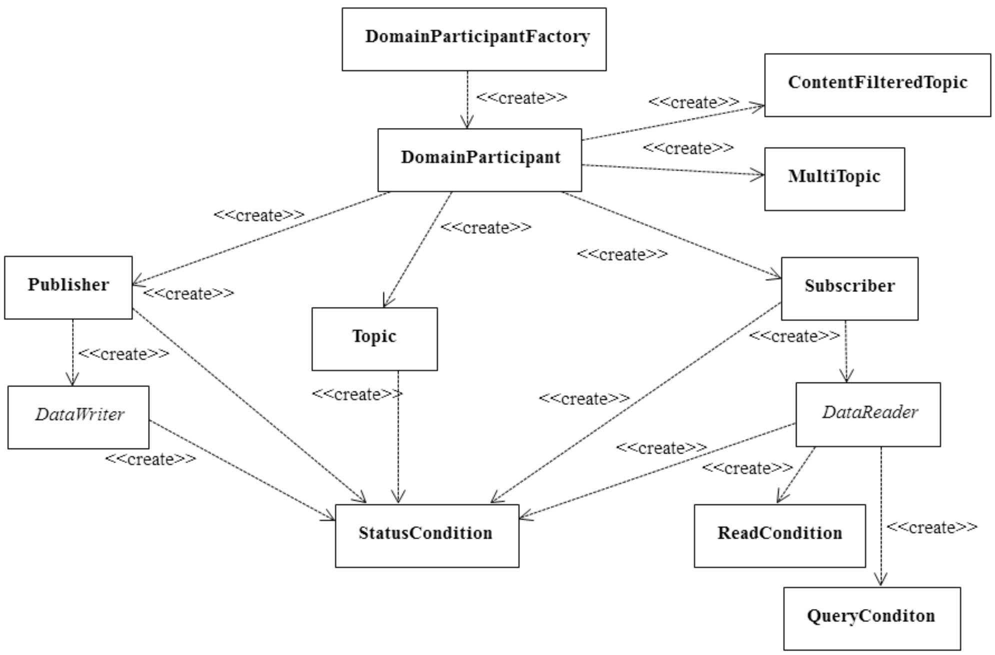


Figure 1.3: Overview of DDS Entities

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

- **MultiTopic**
- **ContentFilteredTopic**
- **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 (available here if you have Internet access). 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.

1.5.3 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.

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 DDS* internal buffers from where the DDS data sample is sent under the criteria set by the *Publisher's* and *DataWriter's* QosPolicies. If there are matched *DataReaders*, then the DDS data sample will have been passed to the physical transport plug-in/device driver by the time that **write()** returns.

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 QosPolicies. A *Listener* is how *Connext DDS* notifies your application of status changes relevant to the *Entity*. The QosPolicies 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 QosPolicies of the *Publisher* and *DataWriter*, data may be buffered to be sent with the data of other *DataWriters* or not sent at all. By default, the data is sent as soon as the *DataWriter's* **write()** function is called.

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

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 (available here if you have Internet access).

Publisher/Subscriber QosPolicies

Please refer to the C API Reference and C++ API Reference for details on supported QosPolicies.

DataWriter QosPolicies

Please refer to the C API Reference and C++ API Reference for details on supported QosPolicies.

1.5.4 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 the *Interoperability* section for more information.

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 asynchronously 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()** operation gives your application a copy of the data and leaves the data in the *DataReader's* receive queue. The *DataReader's* **take()** operation removes data from the receive queue before giving it to your application.

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 asynchronously:
  - 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*.

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 QosPolicies. The *Listener* is how *RTI Connext Micro* notifies your application of status changes relevant to the *Entity*. The QosPolicies 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 QosPolicies 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 QosPolicies, see the Subscribers section in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

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 HTML documentation.

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*.

Subscriber QosPolicies

Please refer to the C API Reference and C++ API Reference for details on supported QosPolicies.

DataReader QosPolicies

Please refer to the C API Reference and C++ API Reference for details on supported QosPolicies.

1.5.5 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.

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.

As seen in *Figure 4.2: Relationship between Applications and DDS Domains*, 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

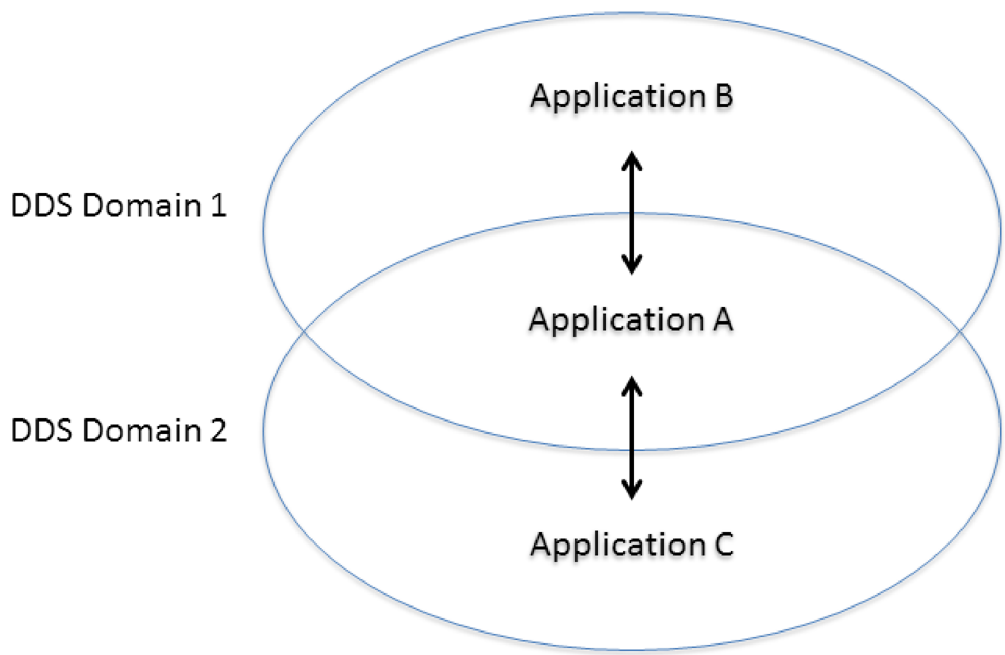


Figure 1.4: Relationship between Applications and DDS Domains
Applications can belong to multiple DDS domains—*A* belongs to DDS domains 1 and 2. Applications in the same DDS domain can communicate with each other, such as *A* and *B*, or *A* and *C*. Applications in different DDS domains, such as *B* and *C*, are not even aware of each other and will not exchange messages.

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 4.3: DDS Domain Module*, 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*).

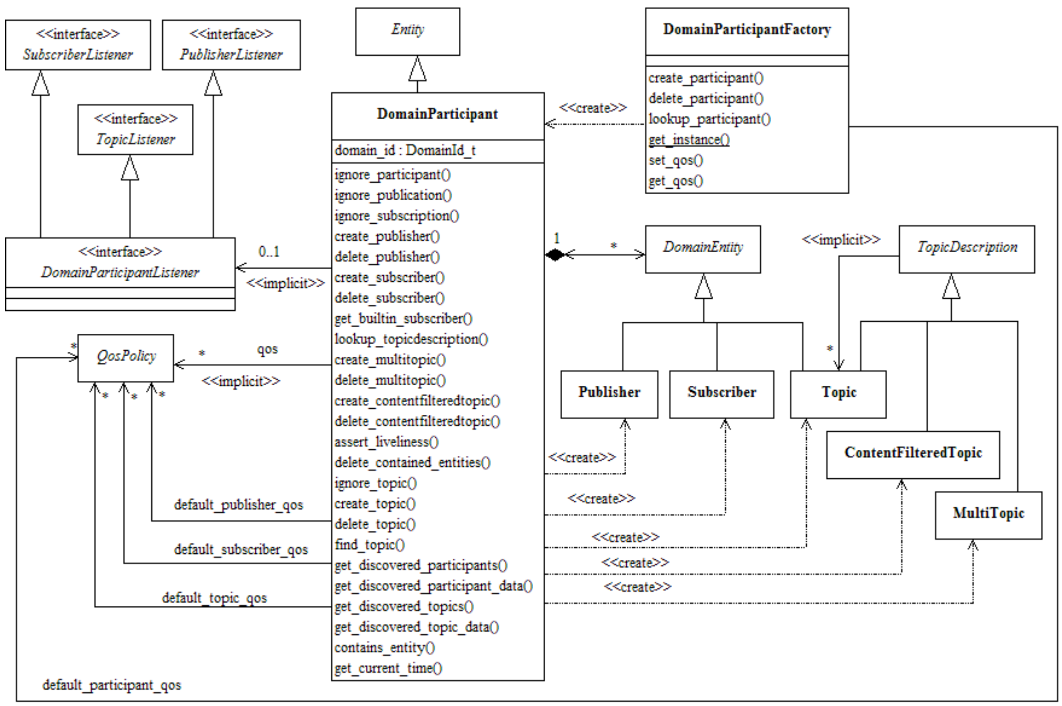


Figure 1.5: DDS Domain Module
Note: MultiTopics are not supported.

Discovery Announcements

Each *DomainParticipant* announces information about itself, such as which locators 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 their own locators, only user-traffic locators can be specified, then the exact same rules apply as for the *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.

1.5.6 Transports

Introduction

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.

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);
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
```

(continues on next page)

(continued from previous page)

```
                                                    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.

Transport Addresses

Address reservation is the process to determine which locators should be used in the discovery announcement. Which transports and addresses to be used is 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.

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 is also not known, thus 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.

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.

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.

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 RTPS_InterfaceFactoryProperty *rtps_property = NULL;

    /* get the Domain Participant factory and registry*/
    factory = DDS_DomainParticipantFactory_get_instance();

    registry =  DDS_DomainParticipantFactory_get_registry
                    (DDS_DomainParticipantFactory_get_instance());


    /* 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,
```

(continues on next page)

(continued from previous page)

```
                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).

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_CalculateChecksum_T calculate_checksum;
} 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 *calculate_checksum* 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.

RTPS_Checksum_T checksum: This is the out parameter of this function. It is a union defined as follows:

```
typedef union RTPS_Checksum
{
        RTI_UINT32 checksum32;
        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* endinaness.
2. The number returned in checksum64 is assumed to be in *host order* endinaness.
3. checksum128 is treated as an octet array.

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,
                                unsigned int buf_length,
                                union RTPS_CrcChecksum *checksum)
{
    RTI_UINT32 crc = 0;
    unsigned char *data = (unsigned char *) buf[0].pointer;
    RTI_UINT32 length = buf[0].length;
    int k;

    UNUSED_ARG(k);
    UNUSED_ARG(context);
    UNUSED_ARG(buf_length);

    for (k = 0; k < length; k++)
```

(continues on next page)

(continued from previous page)

```
    {
        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;
    struct RTPS_InterfaceFactoryProperty *rtps_property = NULL;

    /* Instantiate a RTPS_CrcClass 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
                    (DDS_DomainParticipantFactory_get_instance());


    /* 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;
    }
```

(continues on next page)

(continued from previous page)

```
    /* 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;
    }


    /* 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);
}
```

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.

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. 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.

Reliability and Durability

Because a sample sent over INTRA bypasses the RTPS reliability and DDS durability queue, the Reliability and Durability Qos policies are *not* supported by the INTRA transport. However, by creating all the *DataReaders* before the *DataWriters* durability is not required.

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.

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

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 = ...

    factory = DDS_DomainParticipantFactory_get_instance();

    registry = DDS_DomainParticipantFactory_get_registry(factory);
    if (!RT_Registry_register(
            registry,
```

(continues on next page)

(continued from previous page)

```
            "_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 */
    }
}
```

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.

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 = ...;

/* 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 */
}
```

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*.

Caveats

Leftover shared memory resources

Connext Micro implements the shared memory transport and utilizes shared memory semaphores that can be used conccurently 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 occurs 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.

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 *Generating Examples* for more details).

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.

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():

```c
if (!NDDS_Transport_ZeroCopy_initialize(registry, NULL, NULL))
{
    printf("failed to initialize zero copy\n");
    /* handle error */
}
```

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 provies 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 processesed 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;
}
```

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).

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.

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
  - 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

[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_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

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*.

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

Registering the UDP Transport

The builtin UDP transport is a *Connext Micro* component that is automatically registered when the DDS_DomainParticipantFactory_get_instance() method is called. To change the UDP configuration, it is necessary to first unregister the transport as shown below:

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

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

/* The builtin transport does not return any properties (3rd param) or
 * listener (4th param)
 */
if (!RT_Registry_unregister(registry, "_udp", NULL, NULL))
{
    /* ERROR */
}
```

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 with new parameters.

```
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;
     */
    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 again, using the builtin name
```

(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 */
}
```

It should be noted that the UDP 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.

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.

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 */
}
```

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 65507 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)->
                            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
```

(continues on next page)

(continued from previous page)

```
 * 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 out the interface list (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.

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*
- *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.

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, for example, UNIX. 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
{
    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.
```

(continues on next page)

(continued from previous page)

```
 */

 /*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
{
    /*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.
```

(continues on next page)

(continued from previous page)

```
 */
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
     * 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
```

(continues on next page)

(continued from previous page)

```
 */
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 sucess, 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;
    }

    /* 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;
    }
```

(continues on next page)

(continued from previous page)

```
    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
 * 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);
```

(continues on next page)

(continued from previous page)

```
    /* 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,
                                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
 */
```

(continues on next page)

(continued from previous page)

```
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;
}

/*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
 *
```

(continues on next page)

(continued from previous page)

```
 * \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;
    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)
```

(continues on next page)

(continued from previous page)

```
        {
            *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
 * \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;
    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;
    }

```

(continues on next page)

(continued from previous page)

```
    *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;
        ++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);
```

(continues on next page)

(continued from previous page)

```
    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*
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;
```

(continues on next page)

(continued from previous page)

```
    }

    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,
    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,
```

(continues on next page)

(continued from previous page)

```
                                (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);
    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:
```

(continues on next page)

(continued from previous page)

```
      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;
    }

    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;
    }
```

(continues on next page)

(continued from previous page)

```
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");
    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.
```

(continues on next page)

(continued from previous page)

```
     */
    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;
    }

    /* 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(),
```

(continues on next page)

(continued from previous page)

```
                                &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);
    DDS_StringSeq_set_length(&dp_qos.discovery.enabled_transports,1);

    /* Use network interface 192.168.10.232 for discovery. TO 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);
```

(continues on next page)

(continued from previous page)

```
    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 =
        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)
{
```

(continues on next page)

(continued from previous page)

```
    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;

    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;
    }
```

(continues on next page)

(continued from previous page)

```
    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.

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.

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.

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);
```

(continues on next page)

(continued from previous page)

```
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                            DDS_String_dup("xyz");

/* 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("xyz://");

/* 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("xyz://");
```

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");
```

Datagram UDP Setup

The built-in Datagram transport can support UDP integration.

The registering the built-in Datagram transport for UDP registers differently 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) =
                                                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;
```

(continues on next page)

(continued from previous page)

```
    udp_property->multicast_interface = DDS_String_dup(intf);
}
else
{
    /* Set the mutlicast 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 a 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");

/* _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.

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 1.2: 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.
`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.

continues on next page

Table 1.2 – continued from previous page

Interface Attribute	Description
`bind_address`	Instructs the NETIO_DGRAM interface to listen for messages on the source address.

1.5.7 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.

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).

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.

peer_desc_string

A peer descriptor string of the initial_peers string sequence conveys the interface and address of the locator to which to send, as well as the indices of participants to which to send. For example:

```
DDS_StringSeq_set_maximum(&dp_qos.discovery.initial_peers, 3);
DDS_StringSeq_set_length(&dp_qos.discovery.initial_peers, 3);

*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 0) =
     DDS_String_dup("_udp://239.255.0.1");

*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 1) =
     DDS_String_dup("[1-4]@_udp://10.10.30.101");

*DDS_StringSeq_get_reference(&dp_qos.discovery.initial_peers, 2) =
     DDS_String_dup("[2]@_udp://10.10.30.102");
```

The peer descriptor format is:

```
[index@][interface://]address
```

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.

In the above example, the first peer, "_udp://239.255.0.1," has the default UDPv4 multicast peer locator. Note that there is no [index@] associated with a multicast locator.

The second peer, "[1-4]@_udp://10.10.30.101," has a unicast address. It also has indices in brackets, [1-4]. These represent a range of participant indices, 1 through 4, to which participant discovery messages will be sent.

Lastly, the third peer, "[2]@_udp://10.10.30.102," is a unicast locator to a single participant with index 2.

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, so while DiscoveryQosPolicy_enabled_transports by default includes the DDS default loopback and multicast (239.255.0.1) addresses, initial_peers must be configured to include them.

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.

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 protocol.

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");
    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;
```

(continues on next page)

(continued from previous page)

```
/* 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,
                                        "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 *DataRead*, 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*.

1.5.8 User Discovery Data

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.

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.

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",
```

(continues on next page)

(continued from previous page)

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

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 constructor initializes memory to the maximum value, except for USER_DATA, TOPIC_DATA, and GROUP_DATA, which it sets to an empty sequence. The overloaded constructor accepts a *DomainParticipant* and initializes the memory according to the resource limits, including memory for USER_DATA, TOPIC_DATA, and GROUP_DATA.

For example, to retrieve information on discovered participants:

```c
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 */
}

/* 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 */
   }
}
```

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 1.3. 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 1.3: 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 1.4. 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 1.4: 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 1.5. 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 1.5: DDS_GroupDataQosPolicy

Type	Field Name	Description
DDS_OctetSeq	value	Empty by default

1.5.9 Partitions

Introduction

The PARTITION QoS 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 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 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 1.6 illustrates the concept of PARTITION QoS 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.

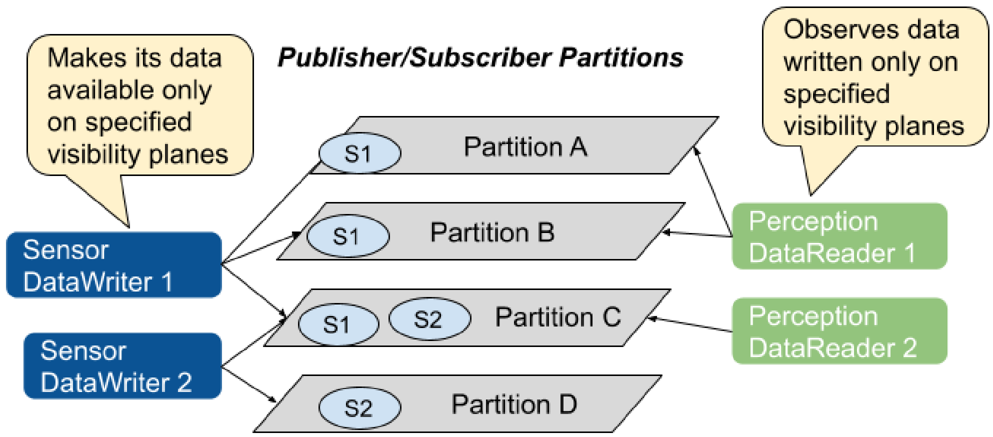


Figure 1.6: Controlling Visibility of Data with PARTITION QoS

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 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 on any of the entities, *DataReaders* 1 and 2 would have received all data samples written by *DataWriters* 1 and 2.

Rules for PARTITION matching

The PARTITION QosPolicy 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, 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 offered by the *DataWriter* is compatible with the QoS requested by the *DataReader*.

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 and does not trigger any listeners or change any status conditions.

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 PARTITON.name element of a different *Entity's* PARTITION QosPolicy.

If a PARTITION QoS only contains regular expressions, then the *Entity* will be assigned automatically to the default partition with the empty string name (""). Thus, a PARTITION QoS that only contains the string `*` matches another *Entity's* PARTITION QoS 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 is shown in Table 1.6. The QosPolicy 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 1.6: DDS_PartitionQosPolicy

Type	Field Name	Description
DDS_StringSeq	name	Empty by default. 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 1.7.

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

Table 1.7: Wild Card Matching

Character	Meaning
,	**NOT SUPPORTED** A , separates a list of alternate patterns. The field string is matched if it matches one or more of the patterns.
/	**NOT SUPPORTED** A / in the pattern string matches a / in the field string. It separates a sequence of mandatory substrings.
?	A ? in the patterns tring 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** 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.

Example

The PARTITION QosPolicy 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 QosPolicy 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)
   {
      ...
   }
```

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 1.8.

Table 1.8: 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" "USA/California/Sunnyvale"	Receive information for Santa Clara or Sunnyvale, California.
(Publisher partition is irrelevant here.)	"USA/California/*" "USA/Nevada/*"	Receive information for California or Nevada.
(Publisher partition is irrelevant here.)	"USA/California/*" "USA/Nevada/Reno" "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 1.9

Table 1.9: 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" "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" "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.

Properties

This QosPolicy cannot be modified at runtime.

Strictly speaking, this QosPolicy 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.

Resource limits

Before this QoS policy can be used, you must configure the following DomainParticipantResource-LimitsQosPolicy fields:

- `max_partitions`: sets the maximum number of partitions for each PARTITION QoS.
- `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 QosPolicy 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 will be reused for PARTITION names that are added, internally deleted, and no longer needed. 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.

1.5.10 Generating Type Support with rtiddsgen

Why Use rtiddsgen?

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

Rather than have users manually implement each new type, *Connext Micro* provides the *rtiddsgen* utility for automatically generating type support code.

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 (available here if you have Internet access).

Generating Type Support

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

- `RTIMEHOME` sets the path of the *Connext Micro* installation directory
- `RTIMEARCH` sets the platform architecture (e.g. i86Linux2.6gcc4.4.5 or i86Win32VS2010)
- `JREHOME` sets the path for a Java JRE

Note that 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.

C

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

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

C++

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

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

Notes on Command-Line Options

In order to target *Connext Micro* when generating code with *rtiddsgen*, the `-micro` option must be specified on the command line.

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

```
> cd $rti_connext_micro_root/rtiddsgen/scripts
> rtiddsgen -micro -help
```

Existing users might notice that that previously available options, `-language microC` and `-language microC++`, have been replaced by `-micro -language C` and `-micro -language C++`, respectively. It is still possible to specify `microC` and `microC++` for backwards compatibility, but users are advised to switch 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(xx) implement creation/intialization and deletion of a single sample and a sequence of samples of the type (or types) defined in the IDL description.
- UserTypePlugin.h and UserTypePlugin.c(xx) implement the pluggable type interface that *Connext Micro* uses to serialize and deserialize the type.
- UserTypeSupport.h and UserTypeSupport.c(xx) define type-specific *DataWriters* and *DataReaders* for user-defined types.

Using custom data-types in Connext Micro Applications

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

```
#include "HelloWorldPlugin.h"

/* ... */

retcode = DDS_DomainParticipant_register_type(application->participant,
                                              "HelloWorld",
                                              HelloWorldTypePlugin_get());
if (retcode != DDS_RETCODE_OK)
{
    /* Log an error */
    goto done;
}

application->topic =
```

(continues on next page)

(continued from previous page)

```
    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;
}
```

See the full HelloWorld examples for further reference.

Customizing generated code

rtiddsgen allows *Connext Micro* users to select whether they 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 provide by two command-line arguments:

- `-reader` generates code for deserializing custom data-types and creating *DataReaders* from them.
- `-writer` generates code for serializing custom data-types and creating *DataWriters* from them.

If neither of these two options are supplied to *rtiddsgen*, they will both be considered active and code for both *DataReaders* and *DataWriters* will be generated. If only one of the two options is supplied to *rtiddsgen*, only that one is enabled. If both options are supplied, both are enabled.

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*.

1.5.11 Threading Model

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*

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.

```
+--------+                                     \
| DDS_C |                                       }  C API
+--------+                                     /

+--------+ +--------+ +-------+ +-------+      \
| DPSE   | | DPDE   | | WHSM | | RHSM |        |
+--------+ +--------+ +-------+ +-------+      |
+--------+ +--------+ +-------+ +-------+ +------+   } Optional components
| LOOP   | | UDP(*)| | RTPS | | DRI   | | DWI |  |  (platform independent)
+--------+ +--------+ +-------+ +-------+ +------+  |
                                               /

+--------+ +--------+ +-------+ +-------+      \  Core Services (always
| REDA   | |  CDR   | | DB    | | RT    |       } present, platform
+--------+ +--------+ +-------+ +-------+      /  independent)

+-----------------------------------+          \
|               OSAPI               |           } Platform dependent module
+-----------------------------------+          /

(*) The UDP transport relies on a BSD socket API
```

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. Thus, the following discussion can only be guaranteed to be true for *Connext Micro* libraries from RTI.

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 = ....

/* 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);

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 = ....
```

(continues on next page)

(continued from previous page)

```
/* 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.

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.

1.5.12 Batching

This section is organized as follows:

- *Overview*
- *Interoperability*
- *Performance*
- *Example Configuration*

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).

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.

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.

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:

```xml
<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:

```xml
<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).

```xml
<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.

1.5.13 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*.

RTPS Checksum

Connext Micro implements checksum validation on a complete RTPS message. A typical RTPS message without a checksum has the following structure:

```
+--------+------------+------------------------+------------+
| Header | Submessage | ...... submessages .... | Submessage |
+--------+------------+------------------------+------------+
```

When the message integrity checking feature is enabled, the structure of the RTPS message changes as illustrated below:

```
+--------+------------+------------+-----------------+------------+
| Header | Checksum   | Submessage | .. submessages ..| Submessage |
+--------+------------+------------+-----------------+------------+
```

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*.

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 the *Participant Discovery and Participant Compatibility* section 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;
```

Participant Discovery and Participant 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.

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.

1.5.14 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:

- *Overview*
- *Configuration of Large Data*
- *Limitations*

Overview

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.

When a sample larger than the transport size is sent, *Connext Micro* splits the sample into fragments and starts sending fragments based on a flow-control algorithm. A bandwidth-allocation parameter on the *DataWriter* and the scheduling rate determine how frequently and how much data can be sent each period.

When a sample is received in multiple fragments, the receiver reassembles each fragment into a complete serialized sample. The serialized data is then deserialized and made available to the user as regular data.

When working with large data, it is important to keep the following in mind:

- Fragmentation is always enabled.
- Fragmentation is per *DataWriter*.
- Flow-control is per *DataWriter*. It is important to keep this in mind since in *RTI Connext Professional* the flow-controller works across all *DataWriters* in the same publisher.
- Fragmentation is on a per sample basis. That is, two samples of the same type may lead to fragmentation of one sample, but not the other. The application is never exposed to fragments.
- It is the *DataWriters* that determine the fragmentation size. Different *DataWriters* can use different fragmentation sizes for the same type.
- All fragments must be received before a sample can be reconstructed. When using best-effort, if a fragment is lost, the entire sample is lost. When using reliability, a fragment that is not received may be resent. If a fragment is no longer available, the entire sample is dropped.
- If one of the DDS **write()** APIs is called too fast when writing large samples, *Connext Micro* may run out of resources. This is because the sample may take a long time to send and resources are not freed until the complete sample has been sent.

It is important to distinguish between the following concepts:

- Fragmentation by *Connext Micro*.
- Fragmentation by an underlying transport, e.g., IP fragmentation when UDP datagrams exceed about 1488 bytes.
- The maximum transmit message size of the sender. This is the maximum size of any payload going over the transport.
- The maximum transport transmit buffer size of the sender. This is the maximum number of bytes that can be stored by the transport.
- The maximum receive message size of a receiver. This is the maximum size of a single payload on a transport.
- The maximum receive buffer size of a receiver. This is the maximum number of bytes that can be received.

Configuration of Large Data

For a general overview of writing large data, please refer to these sections in the RTI Connext DDS Core Libraries User's Manual:

- the ASYNCHRONOUS_PUBLISHER QoSPolicy section (available here if you have Internet access)
- the FlowControllers section (available here if you have Internet access)

NOTE: *Connext Micro* only supports the default FlowController.

Asynchronous publishing is handled by a separate thread that runs at a fixed rate. The rate and properties of this thread can be adjusted in the OSAPI_SystemProperty and the following fields before DomainParticipantFactory_get_instance() is called.

```
struct OSAPI_SystemProperty sys_property = OSAPI_SystemProperty_INITIALIZER;
DDS_DomainParticipantFactory *factory = NULL;

if (!OSAPI_System_get_property(&sys_property))
{
    /* error */
}

sys_property.task_scheduler.thread.stack_size = ....
sys_property.task_scheduler.thread.options = ....
sys_property.task_scheduler.thread.priority = ....
sys_property.task_scheduler.rate = rate in nanosec;

if (!OSAPI_System_set_property(&sys_property))
{
    /* error */
}

factory = DDS_DomainParticipantFactory_get_instance();

....
```

Limitations

The following are known limitations and issues with Large Data support:

- It is not possible to disable fragmentation support.
- The scheduler thread accuracy is based on the operating system.

1.5.15 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 1.7 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.

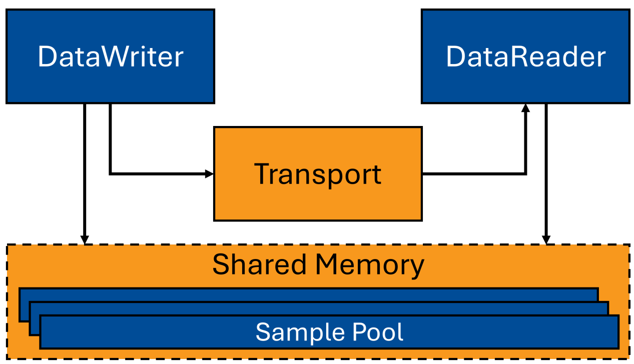


Figure 1.7: Zero Copy Transfer in *Connext Micro*

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*.

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 1.10: 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]	

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()

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.
         */
     }
}
```

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.

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.

1.5.16 FlatData Language Binding

This section is organized as follows:

- *Overview*
- *Getting Started*
- *Further Information*

Overview

RTI Connext Micro supports the FlatData™ language binding in the same manner as *RTI Connext*. However, *Connext Micro* only supports the FlatData language binding for traditional C++ APIs, whereas *RTI Connext* also supports it for the Modern C++ API. The FlatData language binding is not supported for the C language binding.

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
```

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 (available here if you have Internet access).

For details on how to build and read a FlatData sample, see FlatData.

1.5.17 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).

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 *A "Hello, World" Example* 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.

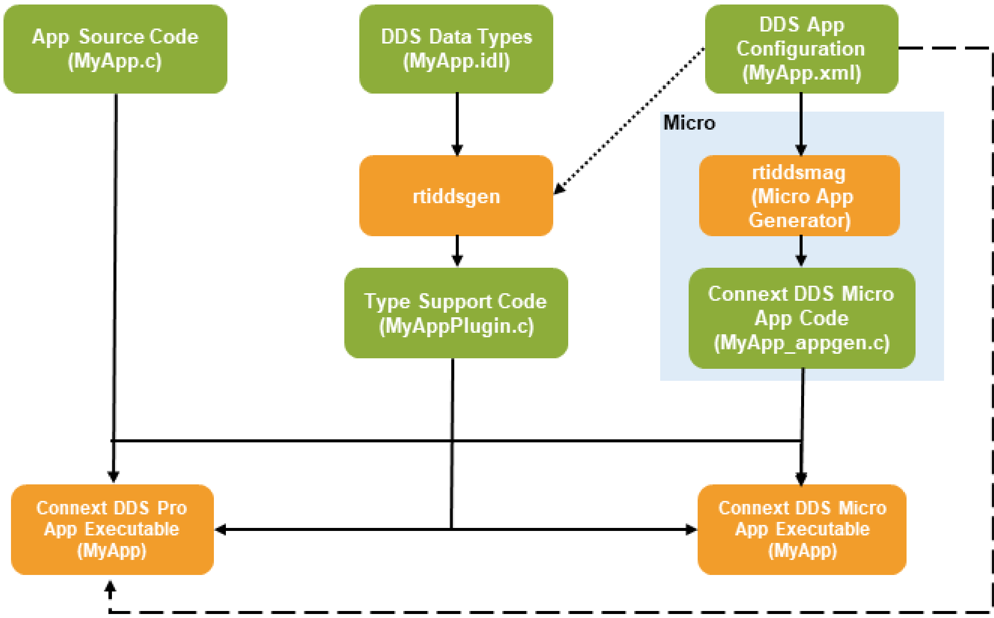
Changes in the XML configuration file require regenerating the C/C++ source code and recompiling the application.

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 and QoS*).
- 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 and QoS*.

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 1.11: 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):
	`struct DDS_TypePluginI *HelloWorldPlugin_get_cpp(void)`
	`{`
	`    return HelloWorldPlugin_get();`
	`}`

> **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 1.12: Command-Line Options for rtiddsmag

Option	Description
`-d <outdir>`	Generates the output in the specified directory. By default, MAG will generate files in the directory where the XML file is located.
`-dontAddLocations`	Use this flag to avoid adding the input file location of fields into the generated files. 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. 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.
`-dontUpdateResourceLimits`	Use this flag to avoid automatically updating the resource limit settings for the DomainParticipantFactory, *DomainParticipants*, *DataReaders*, and *DataWriters*. **Note:** The use of this flag for the DomainParticipantFactory is currently not supported. 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. 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`.
`-dpseName <name>`	Specifies the name used by MAG when registering a DPSE discovery plugin. By default, this name is `dpse`.
`-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.
`-outputFinalQos <QosLibrary::QosProfile>`	Use this flag to display the final values of the specified QoS profile after applying inheritance. Although MAG currently doesn't generate code to set the QoS for *Connext Micro*, using this flag will determine the final values in

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_con` 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.

Creating the Application

Call API to Create DomainParticipant

To create the application that MAG generates from the XML definition, you must call the API create_participant_from_config() to create 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 creation, you can retrieve the defined *Entities* by using the lookup_by_name() operations available in the C API Reference and C++ API Reference.

A "Hello, World" Example

This simple scenario consists of two applications: **HelloWorld_publisher**, which writes the *Topic*, HelloWorldTopic, and **HelloWorld_subscriber**, which subscribes to that *Topic*.

The files for this example are generated using rtiddsmag. The instructions are located in the README files in the directories located at <path to Micro examples>/C/. These directories are named `HelloWorld_mag_dpde`, `HelloWorld_mag_dpse`, `HelloWorld_mag_shared_memory`, and `HelloWorld_mag_static_udp`.

The following examples are generated 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".

- **Domain Participant "HelloWorldDPSEPubDP"**

  This application defines a publisher which uses DPSE discovery.

  The application has one named "HelloWorldDPSEPubDP", one named "HelloWorldDPSEPub", and one named "HelloWorldDPSEDW" which uses topic name "Example HelloWorld" and has RTPS id 100. The application registers one type with name "HelloWorld" and defines one with name "Example HelloWorld" which uses type "HelloWorld".

  The application asserts one remote participant named "HelloWorldDPSESubDP" and one remote subscription with ID 200, type name "HelloWorld", and topic name "Example HelloWorld".

- **Domain Participant "HelloWorldDPSESubDP"**

  This application defines a subscriber which uses DPSE discovery.

  The application has one named "HelloWorldDPSESubDP", one named "HelloWorldDPSESub", and one named "HelloWorldDPSEDR" which uses topic name "Example HelloWorld" and has RTPS id 200. The application registers one type with name "HelloWorld" and defines one with name "Example HelloWorld" which uses the type "HelloWorld".

  The application asserts one remote participant named "HelloWorldDPSEPubDP" and one remote subscription with ID 100, type name "HelloWorld", and topic name "Example HelloWorld".

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.

The file **HelloWorld.xml** contains the XML description of the data type.

Let's examine the type used in this example:

```
<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"/>
```

(continues on next page)

(continued from previous page)

```
        <member name="message" type="string" stringMaxLength="MAX_MSG_LEN"/>
        <member name="count"   type="long"/>
    </struct>
</types>
```

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 the DPDE example.

To generate code with rtiddsgen:

- On a Windows system:

  From your command shell, change directory to `<path to Micro examples>\C\HelloWorld_mag_dpde` and type:

  ```
  <RTIMEHOME>\rtiddsgen\scripts\rtiddsgen.bat -example -exampleTemplate mag/dpde -
  ↪language C HelloWorld.xml
  ```

- On a Linux or macOS system:

  From your command shell, change directory to `<path to Micro examples>/C/HelloWorld_mag_dpde` and type:

  ```
  <RTIMEHOME>/rtiddsgen/scripts/rtiddsgen -example -exampleTemplate mag/dpde -
  ↪language C HelloWorld.xml
  ```

After running *rtiddsgen*, you will see 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:

```
typedef struct HelloWorld
{
   CDR_String sender;
   CDR_String message;
   CDR_Long count;
} HelloWorld;
```

- **HelloWorldPlugin.h** contains the `get_plugin_type()` 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: You can do this step before or after generating Type-Support from the Type definition since the type code doesn't need to exist when running *rtiddsmag*.

To generate code with rtiddsmag:

- On a Windows system:

  From your command shell, change directory to `<path to Micro examples>\C\HelloWorld_mag_dpde` and type:

```
<RTIMEHOME>\rtiddsmag\scripts\rtiddsmag.bat -language C -referencedFile␣
↪HelloWorldQos.xml HelloWorld.xml
```

- On a Linux or macOS system:

  From your command shell, change directory to `<path to Micro examples>/C/HelloWorld_mag_dpde` and type:

```
<RTIMEHOME>/rtiddsmag/scripts/rtiddsmag -language C -referencedFile HelloWorldQos.
↪xml HelloWorld.xml
```

We will examine the content of the generated files in the next section.

Examine the XML Configuration Files and the Generated Code

The 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">
        <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>
```

(continues on next page)

(continued from previous page)

```
            </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"/>
        <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 `get_plugin_type` 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 `get_type_plugin` 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 `get_type_plugin` is generated from that attribute plus the string "TypePlugin_get". Our example has type_ref = "HelloWorld", so the value of `get_type_plugin` will be `HelloWorldTypePlugin_get`.

```
<!-- 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">
      </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] =
{
   {
      "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>
        <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** */
```

(continues on next page)

(continued from previous page)

```
    },
    {
        "_intra", /* NETIO_DEFAULT_INTRA_NAME */
        NULL, /* _intra struct RT_ComponentFactoryProperty** */
        NULL  /* _intra struct RT_ComponentFactoryListener** */
    }
};

struct DPDE_DiscoveryPluginProperty
HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde[1] =
{
    RTI_APP_GEN___dpde__HelloWorldAppLibrary_HelloWorldDPDEPubDP_dpde1
};
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] =
{
    {
        "HelloWorldDPDEDW", /* name */
        1UL, /* multiplicity */
        "HelloWorldTopic", /* topic_name */
        RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_
↪HelloWorldDPDEDW /* writer_qos */
    }
};
const struct APPGEN_PublisherModel
```

(continues on next page)

(continued from previous page)

```
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 struct DPDE_DiscoveryPluginProperty HelloWorldAppLibrary_HelloWorldDPDEPubDP_
↪dpde[1];
extern 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 \
{ \
    "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 */
↪ \
```

(continues on next page)

(continued from previous page)

```
    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>
                    <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>
```

(continues on next page)

(continued from previous page)

```
                <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>
            </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>
```

(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>
                <!-- 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>
                    <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>
```

(continues on next page)

(continued from previous page)

```
                            </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.

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>
```

(continues on next page)

(continued from previous page)

```
    <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>
        <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>
```

(continues on next page)

(continued from previous page)

```
        <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>
        </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>
```

(continues on next page)

(continued from previous page)

```
</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 */ \
    }, \
```

(continues on next page)

(continued from previous page)

```
    {   /* 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 */ \
        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 */ \
```

(continues on next page)

(continued from previous page)

```
        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>
            </heartbeat_period>
        </rtps_reliable_writer>
    </protocol>
    <!-- transports -->
    <unicast>
        <value>
            <element>
                <transports>
                    <element>udpv4</element>
                </transports>
            </element>
        </value>
```

(continues on next page)

(continued from previous page)

```
    </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 */ \
            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 */ \
```

(continues on next page)

(continued from previous page)

```
            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_TRUST_INITIALIZER \
    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>
```

(continues on next page)

(continued from previous page)

```
    </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, \
```

(continues on next page)

(continued from previous page)

```
    {   /* 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 */ \
        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_TRUST_INITIALIZER \
    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*:

- UDP Transport: `udp` + participant_number.
- 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.

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 */ \
```

(continues on next page)

(continued from previous page)

```
        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 */ \
                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
- 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

[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_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

```
struct ZCOPY_NotifMechanismProperty zcopy1_user_property = RTI_APP_GEN___zcopy__
↪HelloWorldAppLibrary_HelloWorldDPDEPubDP_zcopy1_NOTIF_USER_PROPERTY;
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 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

rtiddsmag supports configuring the following properties via the PROPERTY QoS policy for the *DomainParticipant*:

- disable_multicast_bind
- multicast_loopback_disable
- disable_multicast_interface_select

Refer to *UDP Configuration* for more information on these properties.

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`.

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>
                <value>100000000</value>
            </element>
```

(continues on next page)

(continued from previous page)

```
            <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 */ \
    }, \
    DDS_BOOLEAN_FALSE /* is_vendor_specific */ \
}
```

(continues on next page)

(continued from previous page)

```
#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 \
{ \
    ...
    {   /* publish_mode */ \
```

(continues on next page)

(continued from previous page)

```
        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:

```xml
<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
HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_participants[1] =
```

(continues on next page)

(continued from previous page)

```
{
    {
        "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 */ \
            {   /* max_blocking_time */ \
```

(continues on next page)

(continued from previous page)

```
                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 \
        DDS_TRUST_SUBSCRIPTION_DATA_INITIALIZER \
    }, \
    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 \
        DDS_TRUST_PUBLICATION_DATA_INITIALIZER \
    }, \
    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 */ \
```

(continues on next page)

(continued from previous page)

```
    NULL /* custom_flow_controllers */ \

#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 \
        DDS_TRUST_SUBSCRIPTION_DATA_INITIALIZER \
    }, \
    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 */ \
```

(continues on next page)

(continued from previous page)

```
    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 */ \
        }, \
        { \
            { 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 \
        DDS_TRUST_PUBLICATION_DATA_INITIALIZER \
    }, \
    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 \
```

(continues on next page)

(continued from previous page)

```
{ \
    "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 */ \
}
```

Errors Caused by Invalid Configurations and QoS

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:

```xml
<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:

```xml
<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*.

```xml
<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>
```

(continues on next page)

(continued from previous page)

```
    </domain_participant_library>
</dds>
```

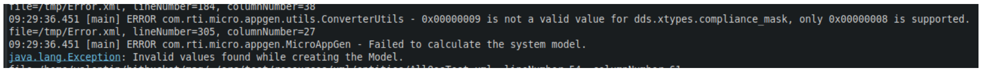
```
07:39:01.248 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Error.xml
07:39:02.069 [main] ERROR com.rti.micro.appgen.utils.ConverterUtils - TRANSIENT_
DURABILITY_QOS is not supported by Micro, only VOLATILE and TRANSIENT_LOCAL are
valid values for the durability kind 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>
        </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.



- **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.

1.5.18 Building Against FACE Conformance Libraries

This section describes how to build *Connext Micro* using the FACE™ conformance test tools.

Requirements

Connext Micro Source Code

The *Connext Micro* source code is available from RTI's Support portal.

FACE Conformance Tools

RTI does not distribute the FACE conformance tools.

CMake

The *Connext Micro* source is distributed 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 UNIX makefiles, Microsoft® Visual Studio® project files, and Xcode.

CMake can be downloaded from https://www.cmake.org.

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 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, it is necessary to port to the required platform. RTI has only tested *Connext Micro* on Linux 2.6 systems with GCC 4.4.5. The complete list of files modified by RTI are included below in source form.

Building the Connext Micro Source

The following instructions show how to built the *Connext Micro* source:

- 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:

```
shell> 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. Press "Done."
- Click "Configure" again. There should not be any red lines. If there are, 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 a 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*:

```
CMAKE_BUILD_TYPE: Debug or blank. If Debug is used, the |me| debug
                  libraries are built.

RTIMICRO_BUILD_API: C or C++
   C   - Include the C API. For FACE, only C is supported.
   C++ - Include the C++ API.

RTIMICRO_BUILD_DISCOVERY_MODULE: Dynamic | Static | Both
   Dynamic - Include the dynamic discovery module.
   Static  - Include the static discovery module.
   Both    - Include both discovery modules.

RTIMICRO_BUILD_LIBRARY_BUILD:
   Single    - Build a single library.
   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.
```

(continues on next page)

(continued from previous page)

```
RTIMICRO_BUILD_LIBRARY_TYPE:
    Static -  Build static libraries.
    Shared -  Build shared libraries.

RTIMICRO_BUILD_LIBRARY_PLATFORM_MODULE: POSIX

RTIMICRO_BUILD_LIBRARY_TARGET_NAME: <target name>
   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
    None            - No compliance required
    General         - Build for compliance with the FACE general profile
    Safety Extended - Build for compliance with the FACE safety extended profile
    Safety          - Build for compliance with the FACE safety profile

RTIMICRO_BUILD_LINK_FACE_GOLDEBLIBS:
    Check if linking against the static FACE conformance test libraries.
    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>**.

1.5.19 Working With Sequences

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.

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.

```
     +---+
  0  | T |
     +---+
  1  | T |
     +---+
  2  | T |
     +---+
       |
       |
     +---+
N-1  | T |
     +---+
```

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().
- 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,0),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)
{
    /* use method */
    *DDSLongSeq_get_reference(&my_sample->long_seq,i) = i;
    snprintf(*DDSStringSeq_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_StringSeq_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.

  Note that it is not allowed to mix the use of calls that pass the max (ends in **_w_max**) and calls that do not. Doing so may cause memory leaks and/or memory corruption.

```
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) = DDS_String_dup("test");
}
```

(continues on next page)

(continued from previous page)

```
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);
```

1.5.20 Debugging

Overview

Connext Micro maintains a log of events occuring 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.

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 that when compiling with RTI_CERT defined, logging is completely removed.

The *Connext Micro* log stores new log entries in a log buffer.

The default buffer size is different for Debug and Release libraries. The Debug libraries are configured to use a much larger buffer than the Release ones. 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_INIITALIZER;

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 in order to fit in the available buffer.

The function used to write the logs can be set during compilation by defining the macro OSAPI_LOG_WRITE_BUFFER. This macro shall have the same parameters as the function prototype **OSAPI_Log_write_buffer_T**.

It is also possible to set this function during runtime by using the function **OSAPI_Log_set_property()**:

```
struct OSAPI_LogProperty prop = OSAPI_LogProperty_INIITALIZER;

OSAPI_Log_get_property(&prop);
prop.write_buffer = <pointer to user defined write function>;
OSAPI_Log_set_property(&prop);
```

A user can install a log handler function to process each new log entry. The handler must conform to the definition OSAPI_LogHandler_T, and it is set by **OSAPI_Log_set_log_handler()**.

When called, the handler has parameters containing the raw log entry and detailed log information (e.g., error code, module, file and function names, line number).

The log handler is called for every new log entry, even when the log buffer is full. An expected use case is redirecting log entries to another logger, such as one native to a particular platform.

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.

Interpreting Log Messages and Error Codes

A log entry in *Connext Micro* has a defined format.

Each entry contains a header with the following information:

- *Length*. The length of the log message, in bytes.
- *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.

Though referred to as an "error" code, it exists for all log kinds (error, warning, info).

The module ID and error code together uniquely identify a log message within *Connext Micro*.

Connext Micro can be configured to provide additional details per log message:

- *Line Number*. The line number of the source file from which the message is logged.
- *Module Name*. The name of the module from which the message is logged.
- *Function Name*. The name of the function from which the message is logged.

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
```

- *X* Extended debug information is present, such as file and line number.
- *E* Exception, the log message has been truncated.
- *T* The log message has a valid timestamp (successful call to OSAPI_System_get_time()).

A log message will need to be interpreted by the user when an error or warning has occurred and its cause needs to be determined, or the user has set a log handler and is processing each log message based on its contents.

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)".

1.6 Platform Notes

1.6.1 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*.

Library types

This section references Platform Independent Libraries (PIL) and Platform Support Libraries (PSL). These are library types that RTI provides in a *Connext Micro* installation. For more information, see *Library types*.

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 1.13: Features by Profile

Feature/Capability	Non-CERT Profile	CERT Profile
Dynamic Participant Discovery	✓	✓
Static Endpoint Discovery	✓	✓
Dynamic Endpoint Discovery	✓	
C++ API	✓	
Shared Memory Transport (SHMEM)	✓[1]	
Zero Copy v1	✓[1]	
Zero Copy v2	✓[1,2]	✓[1,2]
Micro Application Generator (MAG)	✓	

[1] Not supported on FreeRTOS platforms.
[2] Only supported on Linux and QNX platforms.

Supported libraries by platform

The following table shows which *Connext Micro* libraries are supported on each platform (RTI architecture).

Table 1.14: Supported Libraries by Platform

Platform	RTI Architecture	Supported Libraries

Windows 10 x64	x86_64lePEvs2017	`rti_me`
		`rti_me_whsm`
		`rti_me_rhsm`
		`rti_me_discdpse`
		`rti_me_discdpde`
		`rti_me_netiosdm`
		`rti_me_netioshmem`
		`rti_me_appgen`
		`rti_me_cpp`

	x86_64lePEvs2017CERT	`rti_me`

macOS 14 x64	x86_64leMachOclang15.0	`rti_me`
		`rti_me_whsm`
		`rti_me_rhsm`
		`rti_me_discdpse`
		`rti_me_discdpde`
		`rti_me_netiosdm`
		`rti_me_netioshmem`
		`rti_me_appgen`
		`rti_me_cpp`

	x86_64leMa- chOclang15.0CERT	`rti_me`

macOS 14 arm64	armv8leMachOclang15.0	`rti_me`
		`rti_me_whsm`
		`rti_me_rhsm`
		`rti_me_discdpse`
		`rti_me_discdpde`
		`rti_me_netiosdm`
		`rti_me_netioshmem`
		`rti_me_appgen`
		`rti_me_cpp`

	armv8leMa- chOclang15.0CERT	`rti_me`

Ubuntu 22.04 x64	x86_64leElfgcc12.3.0	`rti_me`
		`rti_me_whsm`
		`rti_me_rhsm`
		`rti_me_discdpse`
		`rti_me_discdpde`
		`rti_me_netiosdm`
		`rti_me_netioshmem`
		`rti_me_netiozcopy`
		`rti_me_appgen`
		`rti_me_cpp`

	x86_64leElfgcc12.3.0CERT	`rti_me` `rti_me_netiozcopy`

Ubuntu 18.04 ARMv8	armv8leElfgcc7.3.0	`rti_me`
		`rti_me_whsm`
		`rti_me_rhsm`
		`rti_me_discdpse`
		`rti_me_discdpde`
		`rti_me_netiosdm`

Supported transports by platform

The following table shows which transports are supported on each architecture.

Table 1.15: 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 x64	x86_64leMachOclang15.0 x86_64leMachOclang15.0CERT	✓	✓	✓	✓	
macOS 14 arm64	armv8leMachOclang15.0 armv8leMachOclang15.0CERT	✓	✓	✓	✓	
Ubuntu 22.04 x64	x86_64leElfgcc12.3.0 x86_64leElfgcc12.3.0CERT	✓	✓	✓	✓	✓
Ubuntu 18.04 ARMv8	armv8leElfgcc7.3.0 armv8leElfgcc7.3.0CERT	✓	✓	✓	✓	✓
QNX 7.1 ARMv8	armv8leElfqcc8.3.0	✓	✓	✓	✓	✓
QOS 2.2.1 (QNX OS for Safety)	armv8leElfqcc8.3.0CERT	✓	✓	✓	✓	✓
FreeRTOS 9.0.0 ARMv7E-M	armv7emFreeRTOS9.0gcc7.3.1 armv7emFreeRTOS9.0gcc7.3.1CERT	✓	✓			

1.6.2 FreeRTOS Platforms

The following table shows the currently supported FreeRTOS platforms.

Table 1.16: Supported FreeRTOS Platforms

OS	Version	CPU	Network Stack	Toolchain	Architecture PIL	Architecture PSL
FreeRTOS	9.0.0	ARMv7E-M	LWiP 2.0.0	GCC 7.3.1	armv7emleElfgcc7.3.1 armv7emleElfgcc7.3.1CERT	armv7emleElfgcc7.3.1-FreeRTOS9.0 armv7emleElfgcc7.3.1CERT-FreeRTOS9.0

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 STM32F769I-DISC0 as the reference hardware. This development kit has a STM32F769NIH6 microcontroller with 2 Mbytes of Flash memory and 512 Kbytes of RAM. For a full description, please refer to the microcontroller documentation here.

STMicroelectronics (ST) provides a toolchain called SW4STM32. SW4STM32 is a free multi-OS software environment based on Eclipse, which supports the full range of STM32 microcontrollers and associated boards. SW4STM32 includes the GCC C/C++ compiler, a GDB-based debugger, and an Eclipse-based IDE.

ST also provides STM32CubeF7. STM32CubeF7 gathers all the generic embedded software components required to develop an application on the STM32F7 microcontrollers in a single package.

STM32CubeF7 also includes many examples and demonstration applications. The example, `LwIP_HTTP_Server_Socket_RTOS` is particularly useful as a starting point, since it provides a working FreeRTOS + lwIP configuration.

RTI used the following versions of the different components:

- SW4STM32 version 2.1
- STM32Cube_FW_F7 version V1.7.0
- FreeRTOS version V9.0.0
- lwIP version V2.0.0

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 __LWIPOPTS_H__
#define __LWIPOPTS_H__

#include <limits.h>

#define NO_SYS                  0

/* ---------- Memory options ---------- */
#define MEM_ALIGNMENT           4

#define MEM_SIZE                (50*1024)

#define MEMP_NUM_PBUF           10
```

(continues on next page)

(continued from previous page)

```
#define MEMP_NUM_UDP_PCB        6

#define MEMP_NUM_TCP_PCB        10

#define MEMP_NUM_TCP_PCB_LISTEN 5

#define MEMP_NUM_TCP_SEG        8

#define MEMP_NUM_SYS_TIMEOUT    10


/* ---------- Pbuf options ---------- */
#define PBUF_POOL_SIZE          8

#define PBUF_POOL_BUFSIZE       1524

/* ---------- IPv4 options ---------- */
#define LWIP_IPV4                1

/* ---------- TCP options ---------- */
#define LWIP_TCP                1
#define TCP_TTL                 255

#define TCP_QUEUE_OOSEQ         0

#define TCP_MSS                 (1500 - 40)      /* TCP_MSS = (Ethernet MTU - IP header␣
↪size - TCP header size) */

#define TCP_SND_BUF             (4*TCP_MSS)

#define TCP_SND_QUEUELEN        (2* TCP_SND_BUF/TCP_MSS)

#define TCP_WND                 (2*TCP_MSS)


/* ---------- ICMP options ---------- */
#define LWIP_ICMP               1


/* ---------- DHCP options ---------- */
#define LWIP_DHCP               1


/* ---------- UDP options ---------- */
#define LWIP_UDP                1
#define UDP_TTL                 255


/* ---------- Statistics options ---------- */
#define LWIP_STATS 0
```

(continues on next page)

(continued from previous page)

```
/* ---------- link callback options ---------- */
#define LWIP_NETIF_LINK_CALLBACK        1

/*
   --------------------------------------
   ---------- Checksum options ----------
   --------------------------------------
*/

/*
The STM32F7xx allows computing and verifying checksums by hardware
*/
#define CHECKSUM_BY_HARDWARE


#ifdef CHECKSUM_BY_HARDWARE
/* CHECKSUM_GEN_IP==0: Generate checksums by hardware for outgoing IP packets.*/
#define CHECKSUM_GEN_IP                 0
/* CHECKSUM_GEN_UDP==0: Generate checksums by hardware for outgoing UDP packets.*/
#define CHECKSUM_GEN_UDP                0
/* CHECKSUM_GEN_TCP==0: Generate checksums by hardware for outgoing TCP packets.*/
#define CHECKSUM_GEN_TCP                0
/* CHECKSUM_CHECK_IP==0: Check checksums by hardware for incoming IP packets.*/
#define CHECKSUM_CHECK_IP               0
/* CHECKSUM_CHECK_UDP==0: Check checksums by hardware for incoming UDP packets.*/
#define CHECKSUM_CHECK_UDP              0
/* CHECKSUM_CHECK_TCP==0: Check checksums by hardware for incoming TCP packets.*/
#define CHECKSUM_CHECK_TCP              0
/* CHECKSUM_CHECK_ICMP==0: Check checksums by hardware for incoming ICMP packets.*/
#define CHECKSUM_GEN_ICMP               0
#else
/* CHECKSUM_GEN_IP==1: Generate checksums in software for outgoing IP packets.*/
#define CHECKSUM_GEN_IP                 1
/* CHECKSUM_GEN_UDP==1: Generate checksums in software for outgoing UDP packets.*/
#define CHECKSUM_GEN_UDP                1
/* CHECKSUM_GEN_TCP==1: Generate checksums in software for outgoing TCP packets.*/
#define CHECKSUM_GEN_TCP                1
/* CHECKSUM_CHECK_IP==1: Check checksums in software for incoming IP packets.*/
#define CHECKSUM_CHECK_IP               1
/* CHECKSUM_CHECK_UDP==1: Check checksums in software for incoming UDP packets.*/
#define CHECKSUM_CHECK_UDP              1
/* CHECKSUM_CHECK_TCP==1: Check checksums in software for incoming TCP packets.*/
#define CHECKSUM_CHECK_TCP              1
/* CHECKSUM_CHECK_ICMP==1: Check checksums by hardware for incoming ICMP packets.*/
#define CHECKSUM_GEN_ICMP               1
#endif


/*
   ----------------------------------------------
```

(continues on next page)

(continued from previous page)

```
   ---------- Sequential layer options ----------
   ----------------------------------------------
*/
#define LWIP_NETCONN                    1

/*
   ------------------------------------
   ---------- Socket options ----------
   ------------------------------------
*/
#define LWIP_SOCKET                     1

/*
   ---------------------------------
   ---------- OS options ----------
   ---------------------------------
*/

#define TCPIP_THREAD_NAME              "TCP/IP"
#define TCPIP_THREAD_STACKSIZE          1000
#define TCPIP_MBOX_SIZE                 6
#define DEFAULT_UDP_RECVMBOX_SIZE       2000
#define DEFAULT_TCP_RECVMBOX_SIZE       2000
#define DEFAULT_ACCEPTMBOX_SIZE         2000
#define DEFAULT_THREAD_STACKSIZE        500
#define TCPIP_THREAD_PRIO               osPriorityHigh

/**
* LWIP_SO_RCVBUF==1: Enable SO_RCVBUF processing.
*/
#define LWIP_SO_RCVBUF                  1


/**
* Instruct lwIP to use the errno provided by libc instead of the errno in lwIP.
* If your libc doesn't include errno, you might need to delete these macros.
*/
#undef LWIP_PROVIDE_ERRNO
#define LWIP_ERRNO_INCLUDE "errno.h"

#endif /* __LWIPOPTS_H__ */
```

FreeRTOS

```
#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

/*-----------------------------------------------------------
* Application specific definitions.
*
* These definitions should be adjusted for your application requirements.
```

(continues on next page)

(continued from previous page)

```
*
* 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.
*----------------------------------------------------------*/

/* Ensure stdint is only used by the compiler, and not the assembler. */
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
#include <stdint.h>
extern uint32_t SystemCoreClock;
#endif

#define configUSE_PREEMPTION              1
#define configUSE_IDLE_HOOK               0
#define configUSE_TICK_HOOK               0
#define configCPU_CLOCK_HZ                (SystemCoreClock)
#define configTICK_RATE_HZ                ((TickType_t)1000)
#define configMAX_PRIORITIES              (7)
#define configMINIMAL_STACK_SIZE          ((uint16_t)128)
#define configTOTAL_HEAP_SIZE             ((size_t)(400 * 1024))
#define configMAX_TASK_NAME_LEN           (16)
#define configUSE_TRACE_FACILITY          1
#define configUSE_16_BIT_TICKS            0
#define configIDLE_SHOULD_YIELD           1
#define configUSE_MUTEXES                 1
#define configQUEUE_REGISTRY_SIZE         8
#define configCHECK_FOR_STACK_OVERFLOW    0
#define configUSE_RECURSIVE_MUTEXES       1
#define configUSE_MALLOC_FAILED_HOOK      0
#define configUSE_APPLICATION_TASK_TAG    0
#define configUSE_COUNTING_SEMAPHORES     1
#define configGENERATE_RUN_TIME_STATS     0

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES           0
#define configMAX_CO_ROUTINE_PRIORITIES (2)

/* Software timer definitions. */
#define configUSE_TIMERS             1
#define configTIMER_TASK_PRIORITY    (2)
#define configTIMER_QUEUE_LENGTH     10
#define configTIMER_TASK_STACK_DEPTH 1280

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */
#define INCLUDE_vTaskPrioritySet       1
#define INCLUDE_uxTaskPriorityGet      1
#define INCLUDE_vTaskDelete            1
#define INCLUDE_vTaskCleanUpResources  0
#define INCLUDE_vTaskSuspend           1
```

(continues on next page)

(continued from previous page)

```
#define INCLUDE_vTaskDelayUntil        0
#define INCLUDE_vTaskDelay             1
#define INCLUDE_xTaskGetSchedulerState 1

/* Cortex-M specific definitions. */
#ifdef __NVIC_PRIO_BITS
/* __BVIC_PRIO_BITS will be specified when CMSIS is being used. */
#define configPRIO_BITS         __NVIC_PRIO_BITS
#else
#define configPRIO_BITS         4        /* 15 priority levels */
#endif

#define configLIBRARY_LOWEST_INTERRUPT_PRIORITY   0xf

#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5

#define configKERNEL_INTERRUPT_PRIORITY   ( configLIBRARY_LOWEST_INTERRUPT_PRIORITY <<␣
↪(8 - configPRIO_BITS) )

#define configMAX_SYSCALL_INTERRUPT_PRIORITY  ( configLIBRARY_MAX_SYSCALL_INTERRUPT_
↪PRIORITY << (8 - configPRIO_BITS) )

#define configASSERT( x ) if( ( x ) == 0 ) { taskDISABLE_INTERRUPTS(); for( ;; ); }

#define vPortSVCHandler    SVC_Handler
#define xPortPendSVHandler PendSV_Handler

#endif /* FREERTOS_CONFIG_H */
```

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 1.17: PIL Compiler Flags for FreeRTOS Platforms

Architecture PIL	Library Format	Compiler Flags Used by RTI
armv7emleElfgcc7.3.1	Static Release	-std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -DNDEBUG -O -DRTI_PIL=1
	Static Debug	-std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -O0 -g -DRTI_PIL=1
armv7emleElfgcc7.3.1CERT	Static Release	-DRTI_CERT -std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -DNDEBUG -O -DRTI_PIL=1
	Static Debug	-DRTI_CERT -std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -O0 -g -DRTI_PIL=1

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.
- `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 armv7emleElfgcc7.3.1-FreeRTOS9.0 \
                       -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/armv7emleElfgcc7.3.1-FreeRTOS9.0
```

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 1.18: PSL Compiler Flags for FreeRTOS Platforms

Architecture PSL	Library Format	Compiler Flags Used by RTI
armv7em-leElfgcc7.3.1-FreeRTOS9.0	Static Release	-std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -DNDEBUG -O -DRTI_PSL=1
	Static Debug	-std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -O0 -g -DRTI_PSL=1
armv7em-leElfgcc7.3.1CERT-FreeRTOS9.0	Static Release	-DRTI_CERT -std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -DNDEBUG -O -DRTI_PSL=1
	Static Debug	-DRTI_CERT -std=c99 -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -mcpu=cortex-m7 -mthumb -mfpu=fpv5-sp-d16 -mfloat-abi=hard -ffunction-sections -fdata-sections -O0 -g -DRTI_PSL=1

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.

1.6.3 Linux Platforms

The following table shows the currently supported Linux platforms.

Table 1.19: Supported Linux Platforms

OS	Version	CPU	Network Stack	Toolchain	Architecture PIL	Architecture PSL
Ubuntu	22.04 LTS	x64	OS Default	GCC 12.3.0	x86_64leElfgcc12.3.0 x86_64leElfgcc12.3.0CERT	x86_64leElfgcc12.3.0-Linux5 x86_64leElfgcc12.3.0CERT-Linux5
Ubuntu	18.04 LTS	ARMv8 (64-bit)	OS Default	GCC 7.3.0	armv8leElfgcc7.3.0 armv8leElfgcc7.3.0CERT	armv8leElfgcc7.3.0-Linux4 armv8leElfgcc7.3.0CERT-Linux4

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 1.20: PIL Compiler Flags for Linux Platforms

Architecture PIL	Library Format	Compiler Flags Used by RTI
x86_64leElfgcc12.3.0	Static Release	**C Flags:** -std=c99 -fsigned-char -O2 -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion" -DNDEBUG **C++ Flags:** -Winit-self -fstrict-aliasing -O2 -nostdinc -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DNDEBUG
	Static Debug	**C Flags:** -std=c99 -fsigned-char -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion" -g -DRTI_PIL=1 **C++ Flags:** -Winit-self -fsigned-char -nostdinc -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PIL=1
x86_64leElfgcc12.3.0CERT	Static Release	**C Flags:** -std=c99 -fsigned-char -O2 -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion" -DNDEBUG -DRTI_PIL=1 -DRTI_CERT
	Static Debug	**C Flags:** -std=c99 -fsigned-char -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -g -DRTI_PIL=1 -DRTI_CERT
armv8leElfgcc7.3.0	Static Release	**C Flags:** -std=c99 -fsigned-char -O2 -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion

Warning: The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, in *RTI Connext Micro* 4.1.0, there is 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.

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.

Building Linux applications with Connext Micro

This section describes how RTI built the Platform Support Library (PSL) for Linux 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 1.21: PSL Compiler Flags for Linux Platforms

Architecture PSL	Library Format	Compiler Flags Used by RTI
x86_64leElfgcc12.3.0-Linux5	Static Release	**C Flags:** -std=c99 -Winit-self -O2 -fsigned-char -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DNDEBUG -DRTI_PSL=1 **C++ Flags:** -Winit-self -O2 -fsigned-char -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DNDEBUG -DRTI_PSL=1
	Static Debug	**C Flags:** -std=c99 -fsigned-char -Winit-self -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g-DRTI_PSL=1 **C++ Flags:** -Winit-self -fsigned-char -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PSL=1
x86_64leElfgcc12.3.0CERT-Linux5	Static Release	**C Flags:** -std=c99 -O2 -fsigned-char -Winit-self -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DNDEBUG -DRTI_PSL=1 -DRTI_CERT
	Static Debug	**C Flags:** -std=c99 -fsigned-char -Winit-self -fstrict-aliasing -Wstrict-aliasing=3 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g-DRTI_PSL=1 -DRTI_CERT
armv8leElfgcc7.3.0-Linux4	Static Release	**C Flags:** -std=c99 -O2 -fsigned-char -Winit-self

1.6.4 macOS Platforms

The following table shows the currently supported macOS platforms.

Table 1.22: Supported macOS Platforms

OS	Version	CPU	Network Stack	Toolchain	Architecture PIL	Architecture PSL
macOS	14	x64	OS Default	clang 15.0	x86_64leMachOclang15.0 x86_64leMachOclang15.0CERT	x86_64leMachOclang15.0-Darwin23 x86_64leMachOclang15.0CERT-Darwin23
macOS	14	arm64	OS Default	clang 15.0	armv8leMachOclang15.0 armv8leMachOclang15.0CERT	armv8leMachOclang15.0-Darwin23 armv8leMachOclang15.0CERT-Darwin23

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 1.23: PIL Compiler Flags for macOS Platforms

Architecture PIL	Library Format	Compiler Flags Used by RTI
x86_64leMachOclang15.0	Static Release	**C Flags:** -std=c99 -fsigned-char -O2 -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -Wno-long-long -DNDEBUG -DRTI_PIL=1 **C++ Flags:** -Winit-self -fsigned-char -O2 -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -Wno-long-long -DNDEBUG -DRTI_PIL=1
	Static Debug	**C Flags:** -std=c99 -nostdinc -fsigned-char -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -Wno-long-long -g -DRTI_PIL=1 **C++ Flags:** -Winit-self -fsigned-char -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -Wno-long-long -g -DRTI_PIL=1
x86_64leMachOclang15.0CERT	Static Release	**C Flags:** -std=c99 -fsigned-char -O2 -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -Wno-long-long -DNDEBUG -DRTI_PIL=1 -DRTI_CERT
	Static Debug	**C Flags:** -std=c99 -fsigned-char -nostdinc -Winit-self -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -Wno-long-long -DRTI_PIL=1 -DRTI_CERT
armv8leMachOclang15.0	Static Release	**C Flags:** -std=c99 -fsigned-char -O2

Warning: The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, in *RTI Connext Micro* 4.1.0, there is 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.

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.

Building macOS applications with Connext Micro

This section describes how RTI built the Platform Support Library (PSL) for macOS 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 1.24: PSL Compiler Flags for macOS Platforms

Architecture PSL	Library Format	Compiler Flags Used by RTI
x86_64leMachOclang15.0-Darwin23	Static Release	**C Flags:** -std=c99 -Winit-self -fsigned-char -O2 -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -DNDEBUG -DRTI_PSL=1 **C++ Flags:** -Winit-self -fstrict-aliasing -fsigned-char -O2 -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wdouble-promotion -Wno-strict-prototypes -Wno-c++11-long-long -Wno-long-long -Wno-sign-conversion -DNDEBUG -DRTI_PSL=1
	Static Debug	**C Flags:** -std=c99 -Winit-self -fsigned-char -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -g -DRTI_PSL=1 **C++ Flags:** -Winit-self -fstrict-aliasing -fsigned-char -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wdouble-promotion -Wno-strict-prototypes -Wno-c++11-long-long -Wno-long-long -Wno-sign-conversion -g -DRTI_PSL=1
x86_64leMachOclang15.0CERT-Darwin23	Static Release	**C Flags:** -std=c99 -Winit-self -fsigned-char -O2 -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -DNDEBUG -DRTI_PSL=1 -DRTI_CERT
	Static Debug	**C Flags:** -std=c99 -Winit-self -fsigned-char -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wdouble-promotion -Wno-strict-prototypes -g -DRTI_PSL=1 -DRTI_CERT
armv8leMachOclang15.0-Darwin23	Static Release	**C Flags:** -std=c99 -Winit-self -fsigned-char -O2 -fstrict-aliasing -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align

1.6.5 QNX Platforms

The following table shows the currently supported QNX platforms.

Table 1.25: 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	armv8leElfqcc8.3.0	armv8leElfqcc8.3.0-QNX7.1
QOS (QNX OS for Safety)	2.2.1	ARMv8 (64-bit)	OS Default: io-pkt	qcc_gpp8.3.0	armv8leElfqcc8.3.0CERT	armv8leElfqcc8.3.0CERT-QOS2.2.1

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 1.26: PIL Compiler Flags for QNX Platforms

Architecture PIL	Library Format	Compiler Flags Used by RTI
armv8leElfqcc8.3.0	Static Release	**C Flags:** -nostdinc -fsigned-char -O2 -Vgcc/${version},gcc_ntoaarch64le -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DRTI_PIL=1 **C++ Flags:** -nostdinc -fsigned-char -O2 -Vgcc/${version},gcc_ntoaarch64le -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DRTI_PIL=1
	Static Debug	**C Flags:** -nostdinc -fsigned-char -Vgcc/${version},gcc_ntoaarch64le -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PIL=1 **C++ Flags:** -nostdinc -fsigned-char -Vgcc/${version},gcc_ntoaarch64le -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PIL=1
armv8leElfqcc8.3.0CERT	Static Release	**C Flags:** -nostdinc -fsigned-char -O2 -Vgcc/${version},gcc_ntoaarch64le -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DRTI_PIL=1 -DRTI_CERT
	Static Debug	**C Flags:** -nostdinc -fsigned-char -Vgcc/${version},gcc_ntoaarch64le -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PIL=1 -DRTI_CERT

> **Warning:** The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, in *RTI Connext Micro* 4.1.0, there is 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.

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.

Building QNX applications with Connext Micro

This section describes how RTI built the Platform Support Library (PSL) for QNX 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 1.27: PSL Compiler Flags for QNX Platforms

Architecture PSL	Library Format	Compiler Flags Used by RTI
armv8leElfqcc8.3.0-QNX7.1	Static Release	**C Flags:** -Vgcc/${version},gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DRTI_PSL=1 **C++ Flags:** -Vgcc/${version},gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DRTI_PSL=1
	Static Debug	**C Flags:** -Vgcc/${version},gcc_ntoaarch64le -fsigned-char -Y_gpp -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PSL=1 **C++ Flags:** -Vgcc/${version},gcc_ntoaarch64le -fsigned-char -Y_gpp -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wnon-virtual-dtor -Wcast-align -Wunused -Woverloaded-virtual -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PSL=1
armv8leElfqcc8.3.0CERT-QOS2.2.1	Static Release	**C Flags:** -Vgcc/${version},gcc_ntoaarch64le -fsigned-char -O2 -Y_gpp -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -DRTI_PSL=1 -DRTI_CERT
	Static Debug	**C Flags:** -Vgcc/${version},gcc_ntoaarch64le -fsigned-char -Y_gpp -Winit-self -Wmissing-declarations -Wall -Wextra -Wpedantic -Wshadow -Wcast-align -Wunused -Wconversion -Wsign-conversion -Wlogical-op -Wdouble-promotion -g -DRTI_PSL=1 -DRTI_CERT

1.6.6 Windows Platforms

The following table shows the currently supported Windows platforms.

Table 1.28: Supported Windows Platforms

OS	Version	CPU	Network Stack	Toolchain	Architecture PIL	Architecture PSL
Windows	10	x64	OS Default	Visual Studio 2017	x86_64lePEvs2017 x86_64lePEvs2017CERT	x86_64lePEvs2017-Win10 x86_64lePEvs2017CERT-Win10

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 1.29: PIL Compiler Flags for Windows Platforms

Architecture PIL	Library Format	Compiler Flags Used by RTI
x86_64lePEvs2017	Static Release	**C Flags:** /X /nologo /W3 -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PIL=1 **C++ Flags:** /X -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PIL=1
	Static Debug	**C Flags:** /X /nologo /W3 -DRTI_WIN32 /Zi /Od -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PIL=1 **C++ Flags:** /X -DRTI_WIN32 /Zi /Od -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PIL=1
x86_64lePEvs2017CERT	Static Release	**C Flags:** /X /nologo /W3 -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PIL=1 -DRTI_CERT
	Static Debug	**C Flags:** /X /nologo /W3 -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PIL=1 -DRTI_CERT

Warning: The *RTI Connext Micro* platform independent libraries are built without the standard C header-files. However, in *RTI Connext Micro* 4.1.0, there is 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.

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.

Building Windows applications with Connext Micro

This section describes how RTI built the Platform Support Library (PSL) for Windows 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 1.30: PSL Compiler Flags for Windows Platforms

Architecture PSL	Library Format	Compiler Flags Used by RTI
x86_64lePEvs2017-Win10	Static Release	**C Flags:** /X /nologo /W3 -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PSL=1 **C++ Flags:** /X -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PSL=1
	Static Debug	**C Flags:** /X /nologo /W3 -DRTI_WIN32 /Zi /Od -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PSL=1 **C++ Flags:** /X -DRTI_WIN32 /Zi /Od -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PSL=1
x86_64leP-Evs2017CERT-Win10	Static Release	**C Flags:** /X /nologo /W3 -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PSL=1 -DRTI_CERT
	Static Debug	**C Flags:** /X /nologo /W3 -DRTI_WIN32 -DWIN32_LEAN_AND_MEAN -D_WIN32_WINNT=0x0600 -D_CRT_SECURE_NO_WARNINGS -DRTI_PSL=1 -DRTI_CERT

1.7 Building Connext Micro

1.7.1 Connext Micro Platforms

Connext Micro includes reference Platform Support Libraries (PSL) as both binaries and source code for the target architectures below. The PSL binaries have been tested and validated by RTI; the included source code is identical to the source code used to build the PSL binaries.

- Windows®
- Linux®
- Unix™ (POSIX Compliant)
- macOS® (Darwin)
- QNX®

Refer to the *Supported Platforms and Programming Languages* section for a complete list of all available target architectures.

1.7.2 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 *Connext Micro Platforms* 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.

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 (www.cmake.org) installed. Note that it is not required to use CMake to build *Connext Micro*, and in some cases it may also not be recommended. As a rule of thumb, if *Connext Micro* can be built from the command-line, CMake is recommended.
- be able to run bash shell scripts (Unix type systems) or BAT scripts (Windows machines).

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

Building the PSL

As described in *Library types*, *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 (UNIX) and BAT (Windows) 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 --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`: 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> \
```

(continues on next page)

(continued from previous page)

```
    -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 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 UNIX makefiles were generated:

```
make help
```

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 types* 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 (UNIX) and BAT (Windows) 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 --config Debug --target x86_64lePEvs2017 \
                  -G "Visual Studio 15 2017" --build
```

To build the PSL using the above PIL:

```
RTIMEHOME\resource\scripts\rtime-make --config Debug --target x86_64lePEvs2017-Win10 \
                  -G "Visual Studio 15 2017" --build
```

To build the integrated library:

```
RTIMEHOME\resource\scripts\rtime-make --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 recommendeds 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 *Connext Micro* for an x64 CPU, running Ubuntu 18.04 LTS compiled with gcc 7.3.0.
- `x86_64leElfgcc7.3.0-Linux4`: PSL for *Connext Micro* for an x64 CPU, running Ubuntu 18.04 LTS compiled with gcc 7.3.0.
- `x64Linux4gcc7.3.0`: Integrated library for *Connext Micro* for an x64 CPU, running Ubuntu 18.04 LTS compiled with gcc 7.3.0.
- `x86_64lePEvs2017`: PIL for *Connext Micro* for an x64 CPU, running Windows 10 compiled with Visual Studio 2017.
- `x86_64lePEvs2017-Win10`: PSL for *Connext Micro* for an x64 CPU, running Windows 10 compiled with Visual Studio 2017.
- `x64Win64VS2017`: Integrated library for *Connext Micro* 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 UNIX 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]  \
```

(continues on next page)

(continued from previous page)

```
    -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)

Cross-compiling Connext Micro

Cross-compiling the *Connext Micro* source-code uses the exact same process described in *Building the source*, but requires an additonal 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
```

1.7.3 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.

When compiling *Connext Micro* with compatibility for *Connext Cert*, the following restrictions apply:

- The C++ API is not supported
- Only Dynamic Participant Static Endpoint (DPSE) discovery is available.

- 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()`)
- POSIX®-compliant systems (Linux, macOS, QNX, etc.) and Windows systems are supported
- Only one library, librti_me, is built. While *Connext Micro* consists of different libraries for discovery, reader and writer history, etc, *Connext Cert* consists of only one library
- 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 Zero Copy transport is not supported
- The shared memory transport is 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*

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, x64Linux5gcc12.3.0CERT) are representative of Cert-compatibility profiles. To compile, use the following command:

```
cd <rti_me install directory>
resource/scripts/rtime-make --target x64Linux5gcc12.3.0CERT <other options>
```

The library is generated in the directory `lib/x64Linux5gcc12.3.0CERT`.

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 x64Linux5gcc12.3.0CERT -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 *Generating Examples* for more information about generating examples.

1.8 Working with Connext Micro and Connext Professional

In some cases, it may be necessary to write an application that is compiled against both *Connext Micro* and *Connext Professional*. In general this is not easy to do because *Connext Micro* supports a very limited set of features compared to *Connext Professional*.

However, 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.

Please refer to the *Introduction* for an overview of features and what is supported by *Connext Micro*.

1.8.1 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.

1.8.2 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.

1.8.3 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*.

1.8.4 Standard APIs

APIs that are defined by the standard behave the same between *Connext Micro* and *Connext Professional*.

1.8.5 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.

1.8.6 Interoperability

Connext Micro and *Connext Professional* interoperate on the wire unless noted otherwise.

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.

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*.

1.8.7 Connext Tools

In general, *Connext Micro* is compatible with RTI tools and other products. The following sections provide additional information for each product.

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

Distributed Logger

This product is not supported by *Connext Micro*.

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.

Monitor

This product is not supported by *Connext Micro*.

Recording Service

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.

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.

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.

Wireshark

Wireshark fully supports *Connext Micro*.

Persistence Service

Connext Micro only supports VOLATILE and TRANSIENT_LOCAL durability and does not support the use of Persistence Service.

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*.

1.9 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

1.10 Release Notes

1.10.1 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 types*.

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.

OS	CPU	Compiler	RTI Architecture Abbreviations
Windows® 10	x64	VS 2017	x86_64lePEvs2017 x86_64lePEvs2017CERT
macOS® 14	x64	clang 15.0	x86_64leMachOclang15.0 x86_64leMachOclang15.0CERT
macOS® 14	arm64	clang 15.0	armv8leMachOclang15.0 armv8leMachOclang15.0CERT
Ubuntu® 22.04 LTS	x64	gcc 12.3.0	x86_64leElfgcc12.3.0 x86_64leElfgcc12.3.0CERT
Ubuntu® 18.04 LTS	ARMv8 (64-bit)	gcc 7.3.0	armv8leElfgcc7.3.0 armv8leElfgcc7.3.0CERT
QNX® 7.1	ARMv8 (64-bit)	qcc_gpp8.3.0	armv8leElfqcc8.3.0
QOS 2.2.1 (QNX OS for Safety)	ARMv8 (64-bit)	qcc_gpp8.3.0	armv8leElfqcc8.3.0CERT
FreeRTOS® 9.0.0	Armv7E-M	gcc 7.3.1	armv7emleElfgcc7.3.1 armv7emleElfgcc7.3.1CERT

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.

OS	CPU	Compiler	RTI Architecture Abbreviations
Windows® 10	x64	VS 2017	x86_64lePEvs2017-Win10 x86_64lePEvs2017CERT-Win10
macOS® 14	x64	clang 15.0	x86_64leMachOclang15.0-Darwin23 x86_64leMachOclang15.0CERT-Darwin23
macOS® 14	arm64	clang 15.0	armv8leMachOclang15.0-Darwin23 armv8leMachOclang15.0CERT-Darwin23
Ubuntu® 22.04 LTS	x64	gcc 12.3.0	x86_64leElfgcc12.3.0-Linux5 x86_64leElfgcc12.3.0CERT-Linux5
Ubuntu® 18.04 LTS	ARMv8 (64-bit)	gcc 7.3.0	armv8leElfgcc7.3.0-Linux4 armv8leElfgcc7.3.0CERT-Linux4
QNX® 7.1	ARMv8 (64-bit)	qcc_gpp8.3.0	armv8leElfqcc8.3.0-QNX7.1
QOS 2.2.1 (QNX OS for Safety)	ARMv8 (64-bit)	qcc_gpp8.3.0	armv8leElfqcc8.3.0CERT-QOS2.2.1
FreeRTOS® 9.0.0	Armv7E-M	gcc 7.3.1	armv7emleElfgcc7.3.1-FreeRTOS9.0 armv7emleElfgcc7.3.1CERT-FreeRTOS9.0

Integrated

RTI does not provide integrated libraries for this release.

1.10.2 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 types* 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.

1.10.3 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

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

A "Hello, World" Example 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]

1.10.4 Previous Releases

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 *Generating Examples*.

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]

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 proces; 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>
```

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]

1.10.5 Known Issues

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-5843]

Failure to compile example generated for MAG

When generating an example for Micro Application Generator (MAG), two files are not generated. The two files that should be generated are `<IDL>.xml` and `<IDL_Qos>.xml`.

Please refer to *Application Generation Using XML* for information on how to create these files manually.

[RTI Issue ID MICRO-6801]

Connext Micro does not work if year exceeds 2038

If the date is set beyond the year 2038, *Connext Micro* will not work. This is because the date is reported as a 32 bit unsigned integer; however, *Connext Micro* expects a signed 32-bit value and is therefore interpreting the “wrap around” value as a negative number, causing an error.

[RTI Issue ID MICRO-2295]

Connext Micro does not work with wide-string characters in the network interface name

Connext Micro does not work with wide-string characters (such as Japanese or Chinese characters) in the network interface name.

As a workaround, rename all the system interfaces so that none of them contain wide-string characters.

[RTI Issue ID MICRO-2423]

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]

DDS_DomainParticipantFactory_finalize_instance fails if INTRA transport has been unregistered

The DDS_DomainParticipantFactory_finalize_instance function fails if the INTRA transport has been unregistered previously in the test.

[RTI Issue ID MICRO-4481]

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]

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]

Flow Controllers require RTOS

Flow controllers require an RTOS.

[RTI Issue ID MICRO-6648]

LatencyBudget is not part of the DataReaderQos or DataWriterQos policy

The LatencyBudgetQos policy is not supported and does not appear as part of the DataReader and DataWriter Qos policy documentation. The default value is 0. When creating earliest deadline first (EDF) flow-controllers, the effective scheduling is round-robin.

[RTI Issue ID MICRO-6649]

The Porting Guide is not included in 4.1.0

RTI Connext Micro 4.1.0 has many internal changes from previous versions of *RTI Connext Micro*, and the *RTI Connext Micro* 4.1.0 APIs are not considered stable. Therefore, instructions for porting *RTI Connext Micro* 4.1.0 are not included with *RTI Connext Micro* 4.1.0. If instructions are needed to port *RTI Connext Micro* 4.1.0, please contact support@rti.com.

[RTI Issue ID MICRO-8618]

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 *Platform Notes* for more information.

[RTI Issue ID MICRO-8154]

1.11 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.

1.12 Copyrights

Printed in U.S.A. First printing.
May 2024.

Trademarks

Copy and Use Restrictions

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/

1.13 Third-Party and Open Source Software

This section outlines Real-Time Innovations (RTI) usage of first-level third-party open source software in the *RTI Connext Micro* libraries and utilities.

1.13.1 Connext Micro Libraries

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.

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
```

(continues on next page)

(continued from previous page)

```
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.
```

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


Mark Adler

madler@alumni.caltech.edu
```

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:

1.13.2 Third-Party Software used by the RTIDDSGEN Code-Generation Utility

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. This has a dependency on ANTLR Runtime, StringTemplates v.3.2.1 and ST4 v.4.0.4.
- Version: Release 3.5.2
- Open Source Software License: https://www.antlr3.org/license.html

(continues on next page)

(continued from previous page)

Apache Commons Lang

- Used on Code Generator. We only use the class StringUtils.
- Version 2.6
- Open Source Software License: Apache License Version 2.0 (full text found in the Appendix).

Apache Log4j 2

- This software is distributed with rtiddsmag (Micro Application Generator) as a jar file. The source code is not modified or shipped.
- Version: 2.17.1
- Open Source Software License: https://logging.apache.org/log4j/2.12.x/license.html

Apache Velocity

- This software is included in rtiddsgen (RTI Code Generator). The source code is not modified or shipped.
- Source: http://velocity.apache.org/
- Version: 2.3
- Open Source Software License: The Apache Software License, Version 2.0
  http://velocity.apache.org/engine/devel/license.html

AdoptOpenJDK JRE

- Version 17.0.6 (LTS) Hotspot JVM
- The JRE binaries of the software are distributed with RTI Connext software that uses rtiddsgen (RTI Code Generator). The source code is not modified or shipped.
- https://adoptopenjdk.net/about.html
- Open Source Software Licenses:

Build scripts and other code to produce the binaries, the website and other build infrastructure are licensed under Apache License, Version 2.0. See Appendix. OpenJDK code itself is licensed under GPL v2 with Classpath Exception (GPLv2+CE). See below, in this section.

For Open JDK:

```
The GNU General Public License (GPL)

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share
and change it.  By contrast, the GNU General Public License is intended to
guarantee your freedom to share and change free software--to make sure the
software is free for all its users.  This General Public License applies to
most of the Free Software Foundation's software and to any other program whose
authors commit to using it.  (Some other Free Software Foundation software is
covered by the GNU Library General Public License instead.) You can apply it to
your programs, too.

When we speak of free software, we are referring to freedom, not price.  Our
General Public Licenses are designed to make sure that you have the freedom to
distribute copies of free software (and charge for this service if you wish),
```

(continues on next page)

(continued from previous page)

```
that you receive source code or can get it if you want it, that you can change
the software or use pieces of it in new free programs; and that you know you
can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny
you these rights or to ask you to surrender the rights.  These restrictions
translate to certain responsibilities for you if you distribute copies of the
software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for
a fee, you must give the recipients all the rights that you have.  You must
make sure that they, too, receive or can get the source code.  And you must
show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2)
offer you this license which gives you legal permission to copy, distribute
and/or modify the software.

Also, for each author's protection and ours, we want to make certain that
everyone understands that there is no warranty for this free software.  If the
software is modified by someone else and passed on, we want its recipients to
know that what they have is not the original, so that any problems introduced
by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents.  We
wish to avoid the danger that redistributors of a free program will
individually obtain patent licenses, in effect making the program proprietary.
To prevent this, we have made it clear that any patent must be licensed for
everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification
follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice
placed by the copyright holder saying it may be distributed under the terms of
this General Public License.  The "Program", below, refers to any such program
or work, and a "work based on the Program" means either the Program or any
derivative work under copyright law: that is to say, a work containing the
Program or a portion of it, either verbatim or with modifications and/or
translated into another language.  (Hereinafter, translation is included
without limitation in the term "modification".) Each licensee is addressed as
"you".

Activities other than copying, distribution and modification are not covered by
this License; they are outside its scope.  The act of running the Program is
not restricted, and the output from the Program is covered only if its contents
constitute a work based on the Program (independent of having been made by
running the Program).  Whether that is true depends on what the Program does.
```

(continues on next page)

(continued from previous page)

```
1. You may copy and distribute verbatim copies of the Program's source code as
you receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice and
disclaimer of warranty; keep intact all the notices that refer to this License
and to the absence of any warranty; and give any other recipients of the
Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may
at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus
forming a work based on the Program, and copy and distribute such modifications
or work under the terms of Section 1 above, provided that you also meet all of
these conditions:

    a) You must cause the modified files to carry prominent notices stating
    that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in whole or
    in part contains or is derived from the Program or any part thereof, to be
    licensed as a whole at no charge to all third parties under the terms of
    this License.

    c) If the modified program normally reads commands interactively when run,
    you must cause it, when started running for such interactive use in the
    most ordinary way, to print or display an announcement including an
    appropriate copyright notice and a notice that there is no warranty (or
    else, saying that you provide a warranty) and that users may redistribute
    the program under these conditions, and telling the user how to view a copy
    of this License.  (Exception: if the Program itself is interactive but does
    not normally print such an announcement, your work based on the Program is
    not required to print an announcement.)

These requirements apply to the modified work as a whole.  If identifiable
sections of that work are not derived from the Program, and can be reasonably
considered independent and separate works in themselves, then this License, and
its terms, do not apply to those sections when you distribute them as separate
works.  But when you distribute the same sections as part of a whole which is a
work based on the Program, the distribution of the whole must be on the terms
of this License, whose permissions for other licensees extend to the entire
whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your
rights to work written entirely by you; rather, the intent is to exercise the
right to control the distribution of derivative or collective works based on
the Program.

In addition, mere aggregation of another work not based on the Program with the
Program (or with a work based on the Program) on a volume of a storage or
distribution medium does not bring the other work under the scope of this
License.
```

(continues on next page)

(continued from previous page)

```
3. You may copy and distribute the Program (or a work based on it, under
Section 2) in object code or executable form under the terms of Sections 1 and
2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source
    code, which must be distributed under the terms of Sections 1 and 2 above
    on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to
    give any third party, for a charge no more than your cost of physically
    performing source distribution, a complete machine-readable copy of the
    corresponding source code, to be distributed under the terms of Sections 1
    and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to
    distribute corresponding source code.  (This alternative is allowed only
    for noncommercial distribution and only if you received the program in
    object code or executable form with such an offer, in accord with
    Subsection b above.)

The source code for a work means the preferred form of the work for making
modifications to it.  For an executable work, complete source code means all
the source code for all modules it contains, plus any associated interface
definition files, plus the scripts used to control compilation and installation
of the executable.  However, as a special exception, the source code
distributed need not include anything that is normally distributed (in either
source or binary form) with the major components (compiler, kernel, and so on)
of the operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy
from a designated place, then offering equivalent access to copy the source
code from the same place counts as distribution of the source code, even though
third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as
expressly provided under this License.  Any attempt otherwise to copy, modify,
sublicense or distribute the Program is void, and will automatically terminate
your rights under this License.  However, parties who have received copies, or
rights, from you under this License will not have their licenses terminated so
long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it.
However, nothing else grants you permission to modify or distribute the Program
or its derivative works.  These actions are prohibited by law if you do not
accept this License.  Therefore, by modifying or distributing the Program (or
any work based on the Program), you indicate your acceptance of this License to
do so, and all its terms and conditions for copying, distributing or modifying
the Program or works based on it.
```

(continues on next page)

(continued from previous page)

```
6. Each time you redistribute the Program (or any work based on the Program),
the recipient automatically receives a license from the original licensor to
copy, distribute or modify the Program subject to these terms and conditions.
You may not impose any further restrictions on the recipients' exercise of the
rights granted herein.  You are not responsible for enforcing compliance by
third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues), conditions
are imposed on you (whether by court order, agreement or otherwise) that
contradict the conditions of this License, they do not excuse you from the
conditions of this License.  If you cannot distribute so as to satisfy
simultaneously your obligations under this License and any other pertinent
obligations, then as a consequence you may not distribute the Program at all.
For example, if a patent license would not permit royalty-free redistribution
of the Program by all those who receive copies directly or indirectly through
you, then the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply and
the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or
other property right claims or to contest validity of any such claims; this
section has the sole purpose of protecting the integrity of the free software
distribution system, which is implemented by public license practices.  Many
people have made generous contributions to the wide range of software
distributed through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing to
distribute software through any other system and a licensee cannot impose that
choice.

This section is intended to make thoroughly clear what is believed to be a
consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain
countries either by patents or by copyrighted interfaces, the original
copyright holder who places the Program under this License may add an explicit
geographical distribution limitation excluding those countries, so that
distribution is permitted only in or among countries not thus excluded.  In
such case, this License incorporates the limitation as if written in the body
of this License.

9. The Free Software Foundation may publish revised and/or new versions of the
General Public License from time to time.  Such new versions will be similar in
spirit to the present version, but may differ in detail to address new problems
or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any later
```

(continues on next page)

(continued from previous page)

```
version", you have the option of following the terms and conditions either of
that version or of any later version published by the Free Software Foundation.
If the Program does not specify a version number of this License, you may
choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs
whose distribution conditions are different, write to the author to ask for
permission.  For software which is copyrighted by the Free Software Foundation,
write to the Free Software Foundation; we sometimes make exceptions for this.
Our decision will be guided by the two goals of preserving the free status of
all derivatives of our free software and of promoting the sharing and reuse of
software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR
THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN OTHERWISE
STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE
PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE,
YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL
ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE
PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR
INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA
BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER
OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible
use to the public, the best way to achieve this is to make it free software
which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program.  It is safest to attach
them to the start of each source file to most effectively convey the exclusion
of warranty; and each file should have at least the "copyright" line and a
pointer to where the full notice is found.

    One line to give the program's name and a brief idea of what it does.

    Copyright (C) <year> <name of author>

    This program is free software; you can redistribute it and/or modify it
```

(continues on next page)

(continued from previous page)

```
    under the terms of the GNU General Public License as published by the Free
    Software Foundation; either version 2 of the License, or (at your option)
    any later version.

    This program is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
    more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc., 59
    Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it
starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author Gnomovision comes
    with ABSOLUTELY NO WARRANTY; for details type 'show w'.  This is free
    software, and you are welcome to redistribute it under certain conditions;
    type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may be
called something other than 'show w' and 'show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school,
if any, to sign a "copyright disclaimer" for the program, if necessary.  Here
is a sample; alter the names:

    Yoyodyne, Inc., hereby disclaims all copyright interest in the program
    'Gnomovision' (which makes passes at compilers) written by James Hacker.

    signature of Ty Coon, 1 April 1989

    Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General Public
License instead of this License.


"CLASSPATH" EXCEPTION TO THE GPL

Certain source files distributed by Oracle America and/or its affiliates are
subject to the following clarification and special exception to the GPL, but
only where Oracle has expressly included in the particular source file's header
```

(continues on next page)

(continued from previous page)

```
the words "Oracle designates this particular file as subject to the "Classpath"
exception as provided by Oracle in the LICENSE file that accompanied this code."

    Linking this library statically or dynamically with other modules is making
    a combined work based on this library.  Thus, the terms and conditions of
    the GNU General Public License cover the whole combination.

    As a special exception, the copyright holders of this library give you
    permission to link this library with independent modules to produce an
    executable, regardless of the license terms of these independent modules,
    and to copy and distribute the resulting executable under terms of your
    choice, provided that you also meet, for each linked independent module,
    the terms and conditions of the license of that module.  An independent
    module is a module which is not derived from or based on this library.  If
    you modify this library, you may extend this exception to your version of
    the library, but you are not obligated to do so.  If you do not wish to do
    so, delete this exception statement from your version.




ADDITIONAL INFORMATION ABOUT LICENSING

Certain files distributed by Oracle America, Inc. and/or its affiliates are
subject to the following clarification and special exception to the GPLv2,
based on the GNU Project exception for its Classpath libraries, known as the
GNU Classpath Exception.

Note that Oracle includes multiple, independent programs in this software
package.  Some of those programs are provided under licenses deemed
incompatible with the GPLv2 by the Free Software Foundation and others.
For example, the package includes programs licensed under the Apache
License, Version 2.0 and may include FreeType. Such programs are licensed
to you under their original licenses.

Oracle facilitates your further distribution of this package by adding the
Classpath Exception to the necessary parts of its GPLv2 code, which permits
you to use that code in combination with other independent modules not
licensed under the GPLv2. However, note that this would not permit you to
commingle code under an incompatible license with Oracle's GPLv2 licensed
code by, for example, cutting and pasting such code into a file also
containing Oracle's GPLv2 licensed code and then distributing the result.

Additionally, if you were to remove the Classpath Exception from any of the
files to which it applies and distribute the result, you would likely be
required to license some or all of the other code in that distribution under
the GPLv2 as well, and since the GPLv2 is incompatible with the license terms
of some items included in the distribution by Oracle, removing the Classpath
Exception could therefore effectively compromise your ability to further
distribute the package.
```

(continues on next page)

(continued from previous page)

```
Failing to distribute notices associated with some files may also create
unexpected legal consequences.

Proceed with caution and we recommend that you obtain the advice of a lawyer
skilled in open source matters before removing the Classpath Exception or
making modifications to this package which may subsequently be redistributed
and/or involve the use of third party software.
```

Gson

- Version 2.9.1
- Portions of rtiddsgen (RTI Code Generator) are built using Gson.
- Open Source Software License: The Apache Software License, Version 2.0 (full text found in the *Appendix*)

1.13.3 Micro Application Generator (rtiddsmag)

Apache Commons CLI

- Used in Micro Application Generator.
- Version 1.4
- Open Source Software License: Apache License Version 2.0 (full text found in the Appendix).

Apache Commons Lang

- 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 the Appendix).

Apache Log4j 2

- This software is distributed with rtiddsmag (Micro Application Generator) as a jar file. The source code is not modified or shipped.
- Version: 2.17.1
- Open Source Software License: https://logging.apache.org/log4j/2.12.x/license.html

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: The Apache Software License, Version 2.0 http://velocity.apache.org/engine/devel/license.html

AdoptOpenJDK JRE

- Version 17.0.6 (LTS) Hotspot JVM
- The JRE binaries of the software are distributed with RTI Connext software that uses rtiddsmag (Micro Application Generator). The source code is not modified or shipped.
- https://adoptopenjdk.net/about.html
- Open Source Software Licenses:

Build scripts and other code to produce the binaries, the website and other build infrastructure are licensed under Apache License, Version 2.0. See Appendix. OpenJDK code itself is licensed under GPL v2 with Classpath Exception (GPLv2+CE). See below, in this section.

For Open JDK:

(continues on next page)

(continued from previous page)

```
that you receive source code or can get it if you want it, that you can change
the software or use pieces of it in new free programs; and that you know you
can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny
you these rights or to ask you to surrender the rights.  These restrictions
translate to certain responsibilities for you if you distribute copies of the
software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for
a fee, you must give the recipients all the rights that you have.  You must
make sure that they, too, receive or can get the source code.  And you must
show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2)
offer you this license which gives you legal permission to copy, distribute
and/or modify the software.

Also, for each author's protection and ours, we want to make certain that
everyone understands that there is no warranty for this free software.  If the
software is modified by someone else and passed on, we want its recipients to
know that what they have is not the original, so that any problems introduced
by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents.  We
wish to avoid the danger that redistributors of a free program will
individually obtain patent licenses, in effect making the program proprietary.
To prevent this, we have made it clear that any patent must be licensed for
everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification
follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice
placed by the copyright holder saying it may be distributed under the terms of
this General Public License.  The "Program", below, refers to any such program
or work, and a "work based on the Program" means either the Program or any
derivative work under copyright law: that is to say, a work containing the
Program or a portion of it, either verbatim or with modifications and/or
translated into another language.  (Hereinafter, translation is included
without limitation in the term "modification".) Each licensee is addressed as
"you".

Activities other than copying, distribution and modification are not covered by
this License; they are outside its scope.  The act of running the Program is
not restricted, and the output from the Program is covered only if its contents
constitute a work based on the Program (independent of having been made by
running the Program).  Whether that is true depends on what the Program does.
```

(continues on next page)

(continued from previous page)

```
1. You may copy and distribute verbatim copies of the Program's source code as
you receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice and
disclaimer of warranty; keep intact all the notices that refer to this License
and to the absence of any warranty; and give any other recipients of the
Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may
at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus
forming a work based on the Program, and copy and distribute such modifications
or work under the terms of Section 1 above, provided that you also meet all of
these conditions:

    a) You must cause the modified files to carry prominent notices stating
    that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in whole or
    in part contains or is derived from the Program or any part thereof, to be
    licensed as a whole at no charge to all third parties under the terms of
    this License.

    c) If the modified program normally reads commands interactively when run,
    you must cause it, when started running for such interactive use in the
    most ordinary way, to print or display an announcement including an
    appropriate copyright notice and a notice that there is no warranty (or
    else, saying that you provide a warranty) and that users may redistribute
    the program under these conditions, and telling the user how to view a copy
    of this License.  (Exception: if the Program itself is interactive but does
    not normally print such an announcement, your work based on the Program is
    not required to print an announcement.)

These requirements apply to the modified work as a whole.  If identifiable
sections of that work are not derived from the Program, and can be reasonably
considered independent and separate works in themselves, then this License, and
its terms, do not apply to those sections when you distribute them as separate
works.  But when you distribute the same sections as part of a whole which is a
work based on the Program, the distribution of the whole must be on the terms
of this License, whose permissions for other licensees extend to the entire
whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your
rights to work written entirely by you; rather, the intent is to exercise the
right to control the distribution of derivative or collective works based on
the Program.

In addition, mere aggregation of another work not based on the Program with the
Program (or with a work based on the Program) on a volume of a storage or
distribution medium does not bring the other work under the scope of this
License.
```

(continues on next page)

(continued from previous page)

```
3. You may copy and distribute the Program (or a work based on it, under
Section 2) in object code or executable form under the terms of Sections 1 and
2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source
    code, which must be distributed under the terms of Sections 1 and 2 above
    on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to
    give any third party, for a charge no more than your cost of physically
    performing source distribution, a complete machine-readable copy of the
    corresponding source code, to be distributed under the terms of Sections 1
    and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to
    distribute corresponding source code.  (This alternative is allowed only
    for noncommercial distribution and only if you received the program in
    object code or executable form with such an offer, in accord with
    Subsection b above.)

The source code for a work means the preferred form of the work for making
modifications to it.  For an executable work, complete source code means all
the source code for all modules it contains, plus any associated interface
definition files, plus the scripts used to control compilation and installation
of the executable.  However, as a special exception, the source code
distributed need not include anything that is normally distributed (in either
source or binary form) with the major components (compiler, kernel, and so on)
of the operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy
from a designated place, then offering equivalent access to copy the source
code from the same place counts as distribution of the source code, even though
third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as
expressly provided under this License.  Any attempt otherwise to copy, modify,
sublicense or distribute the Program is void, and will automatically terminate
your rights under this License.  However, parties who have received copies, or
rights, from you under this License will not have their licenses terminated so
long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it.
However, nothing else grants you permission to modify or distribute the Program
or its derivative works.  These actions are prohibited by law if you do not
accept this License.  Therefore, by modifying or distributing the Program (or
any work based on the Program), you indicate your acceptance of this License to
do so, and all its terms and conditions for copying, distributing or modifying
the Program or works based on it.
```

(continues on next page)

(continued from previous page)

```
6. Each time you redistribute the Program (or any work based on the Program),
the recipient automatically receives a license from the original licensor to
copy, distribute or modify the Program subject to these terms and conditions.
You may not impose any further restrictions on the recipients' exercise of the
rights granted herein.  You are not responsible for enforcing compliance by
third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues), conditions
are imposed on you (whether by court order, agreement or otherwise) that
contradict the conditions of this License, they do not excuse you from the
conditions of this License.  If you cannot distribute so as to satisfy
simultaneously your obligations under this License and any other pertinent
obligations, then as a consequence you may not distribute the Program at all.
For example, if a patent license would not permit royalty-free redistribution
of the Program by all those who receive copies directly or indirectly through
you, then the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply and
the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or
other property right claims or to contest validity of any such claims; this
section has the sole purpose of protecting the integrity of the free software
distribution system, which is implemented by public license practices.  Many
people have made generous contributions to the wide range of software
distributed through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing to
distribute software through any other system and a licensee cannot impose that
choice.

This section is intended to make thoroughly clear what is believed to be a
consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain
countries either by patents or by copyrighted interfaces, the original
copyright holder who places the Program under this License may add an explicit
geographical distribution limitation excluding those countries, so that
distribution is permitted only in or among countries not thus excluded.  In
such case, this License incorporates the limitation as if written in the body
of this License.

9. The Free Software Foundation may publish revised and/or new versions of the
General Public License from time to time.  Such new versions will be similar in
spirit to the present version, but may differ in detail to address new problems
or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any later
```

(continues on next page)

(continued from previous page)

```
version", you have the option of following the terms and conditions either of
that version or of any later version published by the Free Software Foundation.
If the Program does not specify a version number of this License, you may
choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs
whose distribution conditions are different, write to the author to ask for
permission.  For software which is copyrighted by the Free Software Foundation,
write to the Free Software Foundation; we sometimes make exceptions for this.
Our decision will be guided by the two goals of preserving the free status of
all derivatives of our free software and of promoting the sharing and reuse of
software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR
THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN OTHERWISE
STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE
PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE,
YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL
ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE
PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR
INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA
BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER
OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible
use to the public, the best way to achieve this is to make it free software
which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program.  It is safest to attach
them to the start of each source file to most effectively convey the exclusion
of warranty; and each file should have at least the "copyright" line and a
pointer to where the full notice is found.

    One line to give the program's name and a brief idea of what it does.

    Copyright (C) <year> <name of author>

    This program is free software; you can redistribute it and/or modify it
```

(continues on next page)

(continued from previous page)

```
    under the terms of the GNU General Public License as published by the Free
    Software Foundation; either version 2 of the License, or (at your option)
    any later version.

    This program is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
    more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc., 59
    Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it
starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author Gnomovision comes
    with ABSOLUTELY NO WARRANTY; for details type 'show w'.  This is free
    software, and you are welcome to redistribute it under certain conditions;
    type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may be
called something other than 'show w' and 'show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school,
if any, to sign a "copyright disclaimer" for the program, if necessary.  Here
is a sample; alter the names:

    Yoyodyne, Inc., hereby disclaims all copyright interest in the program
    'Gnomovision' (which makes passes at compilers) written by James Hacker.

    signature of Ty Coon, 1 April 1989

    Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General Public
License instead of this License.


"CLASSPATH" EXCEPTION TO THE GPL

Certain source files distributed by Oracle America and/or its affiliates are
subject to the following clarification and special exception to the GPL, but
only where Oracle has expressly included in the particular source file's header
```

(continues on next page)

(continued from previous page)

```
the words "Oracle designates this particular file as subject to the "Classpath"
exception as provided by Oracle in the LICENSE file that accompanied this code."

    Linking this library statically or dynamically with other modules is making
    a combined work based on this library.  Thus, the terms and conditions of
    the GNU General Public License cover the whole combination.

    As a special exception, the copyright holders of this library give you
    permission to link this library with independent modules to produce an
    executable, regardless of the license terms of these independent modules,
    and to copy and distribute the resulting executable under terms of your
    choice, provided that you also meet, for each linked independent module,
    the terms and conditions of the license of that module.  An independent
    module is a module which is not derived from or based on this library.  If
    you modify this library, you may extend this exception to your version of
    the library, but you are not obligated to do so.  If you do not wish to do
    so, delete this exception statement from your version.




ADDITIONAL INFORMATION ABOUT LICENSING

Certain files distributed by Oracle America, Inc. and/or its affiliates are
subject to the following clarification and special exception to the GPLv2,
based on the GNU Project exception for its Classpath libraries, known as the
GNU Classpath Exception.

Note that Oracle includes multiple, independent programs in this software
package.  Some of those programs are provided under licenses deemed
incompatible with the GPLv2 by the Free Software Foundation and others.
For example, the package includes programs licensed under the Apache
License, Version 2.0 and may include FreeType. Such programs are licensed
to you under their original licenses.

Oracle facilitates your further distribution of this package by adding the
Classpath Exception to the necessary parts of its GPLv2 code, which permits
you to use that code in combination with other independent modules not
licensed under the GPLv2. However, note that this would not permit you to
commingle code under an incompatible license with Oracle's GPLv2 licensed
code by, for example, cutting and pasting such code into a file also
containing Oracle's GPLv2 licensed code and then distributing the result.

Additionally, if you were to remove the Classpath Exception from any of the
files to which it applies and distribute the result, you would likely be
required to license some or all of the other code in that distribution under
the GPLv2 as well, and since the GPLv2 is incompatible with the license terms
of some items included in the distribution by Oracle, removing the Classpath
Exception could therefore effectively compromise your ability to further
distribute the package.
```

(continues on next page)

(continued from previous page)

```
Failing to distribute notices associated with some files may also create
unexpected legal consequences.

Proceed with caution and we recommend that you obtain the advice of a lawyer
skilled in open source matters before removing the Classpath Exception or
making modifications to this package which may subsequently be redistributed
and/or involve the use of third party software.
```

Extended StAX API

- Version 1.8
- Open Source Software Licenses: https://www.eclipse.org/org/documents/edl-v10.php

Fast Infoset

- Version 1.2.15
- Open Source Software Licenses: Apache License Version 2.0 (full text found in the Appendix).

Istack Common Utility Code Runtime

- Version 3.0.7
- Open Source Software Licenses: https://javaee.github.io/glassfish/LICENSE

JavaBeans Activation Framework API

- Version 1.2.0
- Open Source Software Licenses: https://oss.oracle.com/licenses/CDDL+GPL-1.1

Javax Annotation API

- Version 1.3.2
- Open Source Software Licenses: https://github.com/javaee/javax.annotation/blob/master/LICENSE

JAXB API

- Version 2.3.1
- Open Source Software Licenses: https://javaee.github.io/jaxb-v2/LICENSE

JAXB Runtime

- Version 2.3.1
- Open Source Software Licenses: https://javaee.github.io/jaxb-v2/LICENSE

Simple Logging Facade for Java (SLF4J)

- Version 1.7.35
- This software is included in rtiddsmag (Micro Application Generator).
- Open Source Software Licenses: https://www.slf4j.org/license.html

TXW2

- Version 2.3.1
- Open Source Software Licenses: https://oss.oracle.com/licenses/CDDL+GPL-1.1

1.13.4 Appendix – Open Source Software Licenses

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
      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
```

(continues on next page)

(continued from previous page)

```
   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,
   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:
```

(continues on next page)

(continued from previous page)

```
    (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
    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
```

(continues on next page)

(continued from previous page)

```
   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

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
```

(continues on next page)

(continued from previous page)

```
        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.
```

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

```
The GNU General Public License (GPL)

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share
and change it.  By contrast, the GNU General Public License is intended to
guarantee your freedom to share and change free software--to make sure the
software is free for all its users.  This General Public License applies to
most of the Free Software Foundation's software and to any other program whose
authors commit to using it.  (Some other Free Software Foundation software is
covered by the GNU Library General Public License instead.) You can apply it to
your programs, too.

When we speak of free software, we are referring to freedom, not price.  Our
General Public Licenses are designed to make sure that you have the freedom to
distribute copies of free software (and charge for this service if you wish),
that you receive source code or can get it if you want it, that you can change
the software or use pieces of it in new free programs; and that you know you
can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny
you these rights or to ask you to surrender the rights.  These restrictions
translate to certain responsibilities for you if you distribute copies of the
software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for
a fee, you must give the recipients all the rights that you have.  You must
make sure that they, too, receive or can get the source code.  And you must
show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2)
offer you this license which gives you legal permission to copy, distribute
```

(continues on next page)

(continued from previous page)

```
and/or modify the software.

Also, for each author's protection and ours, we want to make certain that
everyone understands that there is no warranty for this free software.  If the
software is modified by someone else and passed on, we want its recipients to
know that what they have is not the original, so that any problems introduced
by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents.  We
wish to avoid the danger that redistributors of a free program will
individually obtain patent licenses, in effect making the program proprietary.
To prevent this, we have made it clear that any patent must be licensed for
everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification
follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice
placed by the copyright holder saying it may be distributed under the terms of
this General Public License.  The "Program", below, refers to any such program
or work, and a "work based on the Program" means either the Program or any
derivative work under copyright law: that is to say, a work containing the
Program or a portion of it, either verbatim or with modifications and/or
translated into another language.  (Hereinafter, translation is included
without limitation in the term "modification".) Each licensee is addressed as
"you".

Activities other than copying, distribution and modification are not covered by
this License; they are outside its scope.  The act of running the Program is
not restricted, and the output from the Program is covered only if its contents
constitute a work based on the Program (independent of having been made by
running the Program).  Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as
you receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice and
disclaimer of warranty; keep intact all the notices that refer to this License
and to the absence of any warranty; and give any other recipients of the
Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may
at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus
forming a work based on the Program, and copy and distribute such modifications
or work under the terms of Section 1 above, provided that you also meet all of
these conditions:

    a) You must cause the modified files to carry prominent notices stating
```

(continues on next page)

(continued from previous page)

```
    that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in whole or
    in part contains or is derived from the Program or any part thereof, to be
    licensed as a whole at no charge to all third parties under the terms of
    this License.

    c) If the modified program normally reads commands interactively when run,
    you must cause it, when started running for such interactive use in the
    most ordinary way, to print or display an announcement including an
    appropriate copyright notice and a notice that there is no warranty (or
    else, saying that you provide a warranty) and that users may redistribute
    the program under these conditions, and telling the user how to view a copy
    of this License.  (Exception: if the Program itself is interactive but does
    not normally print such an announcement, your work based on the Program is
    not required to print an announcement.)

These requirements apply to the modified work as a whole.  If identifiable
sections of that work are not derived from the Program, and can be reasonably
considered independent and separate works in themselves, then this License, and
its terms, do not apply to those sections when you distribute them as separate
works.  But when you distribute the same sections as part of a whole which is a
work based on the Program, the distribution of the whole must be on the terms
of this License, whose permissions for other licensees extend to the entire
whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your
rights to work written entirely by you; rather, the intent is to exercise the
right to control the distribution of derivative or collective works based on
the Program.

In addition, mere aggregation of another work not based on the Program with the
Program (or with a work based on the Program) on a volume of a storage or
distribution medium does not bring the other work under the scope of this
License.

3. You may copy and distribute the Program (or a work based on it, under
Section 2) in object code or executable form under the terms of Sections 1 and
2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source
    code, which must be distributed under the terms of Sections 1 and 2 above
    on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to
    give any third party, for a charge no more than your cost of physically
    performing source distribution, a complete machine-readable copy of the
    corresponding source code, to be distributed under the terms of Sections 1
    and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to
```

(continues on next page)

(continued from previous page)

```
    distribute corresponding source code.  (This alternative is allowed only
    for noncommercial distribution and only if you received the program in
    object code or executable form with such an offer, in accord with
    Subsection b above.)

The source code for a work means the preferred form of the work for making
modifications to it.  For an executable work, complete source code means all
the source code for all modules it contains, plus any associated interface
definition files, plus the scripts used to control compilation and installation
of the executable.  However, as a special exception, the source code
distributed need not include anything that is normally distributed (in either
source or binary form) with the major components (compiler, kernel, and so on)
of the operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy
from a designated place, then offering equivalent access to copy the source
code from the same place counts as distribution of the source code, even though
third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as
expressly provided under this License.  Any attempt otherwise to copy, modify,
sublicense or distribute the Program is void, and will automatically terminate
your rights under this License.  However, parties who have received copies, or
rights, from you under this License will not have their licenses terminated so
long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it.
However, nothing else grants you permission to modify or distribute the Program
or its derivative works.  These actions are prohibited by law if you do not
accept this License.  Therefore, by modifying or distributing the Program (or
any work based on the Program), you indicate your acceptance of this License to
do so, and all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program),
the recipient automatically receives a license from the original licensor to
copy, distribute or modify the Program subject to these terms and conditions.
You may not impose any further restrictions on the recipients' exercise of the
rights granted herein.  You are not responsible for enforcing compliance by
third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues), conditions
are imposed on you (whether by court order, agreement or otherwise) that
contradict the conditions of this License, they do not excuse you from the
conditions of this License.  If you cannot distribute so as to satisfy
simultaneously your obligations under this License and any other pertinent
obligations, then as a consequence you may not distribute the Program at all.
For example, if a patent license would not permit royalty-free redistribution
of the Program by all those who receive copies directly or indirectly through
```

(continues on next page)

(continued from previous page)

```
you, then the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply and
the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or
other property right claims or to contest validity of any such claims; this
section has the sole purpose of protecting the integrity of the free software
distribution system, which is implemented by public license practices.  Many
people have made generous contributions to the wide range of software
distributed through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing to
distribute software through any other system and a licensee cannot impose that
choice.

This section is intended to make thoroughly clear what is believed to be a
consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain
countries either by patents or by copyrighted interfaces, the original
copyright holder who places the Program under this License may add an explicit
geographical distribution limitation excluding those countries, so that
distribution is permitted only in or among countries not thus excluded.  In
such case, this License incorporates the limitation as if written in the body
of this License.

9. The Free Software Foundation may publish revised and/or new versions of the
General Public License from time to time.  Such new versions will be similar in
spirit to the present version, but may differ in detail to address new problems
or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any later
version", you have the option of following the terms and conditions either of
that version or of any later version published by the Free Software Foundation.
If the Program does not specify a version number of this License, you may
choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs
whose distribution conditions are different, write to the author to ask for
permission.  For software which is copyrighted by the Free Software Foundation,
write to the Free Software Foundation; we sometimes make exceptions for this.
Our decision will be guided by the two goals of preserving the free status of
all derivatives of our free software and of promoting the sharing and reuse of
software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR
```

(continues on next page)

(continued from previous page)

```
THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN OTHERWISE
STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE
PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE,
YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL
ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE
PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR
INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA
BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER
OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible
use to the public, the best way to achieve this is to make it free software
which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program.  It is safest to attach
them to the start of each source file to most effectively convey the exclusion
of warranty; and each file should have at least the "copyright" line and a
pointer to where the full notice is found.

    One line to give the program's name and a brief idea of what it does.

    Copyright (C) <year> <name of author>

    This program is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the Free
    Software Foundation; either version 2 of the License, or (at your option)
    any later version.

    This program is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
    more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc., 59
    Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it
```

(continues on next page)

(continued from previous page)

```
starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author Gnomovision comes
    with ABSOLUTELY NO WARRANTY; for details type 'show w'.  This is free
    software, and you are welcome to redistribute it under certain conditions;
    type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may be
called something other than 'show w' and 'show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school,
if any, to sign a "copyright disclaimer" for the program, if necessary.  Here
is a sample; alter the names:

    Yoyodyne, Inc., hereby disclaims all copyright interest in the program
    'Gnomovision' (which makes passes at compilers) written by James Hacker.

    signature of Ty Coon, 1 April 1989

    Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General Public
License instead of this License.


"CLASSPATH" EXCEPTION TO THE GPL

Certain source files distributed by Oracle America and/or its affiliates are
subject to the following clarification and special exception to the GPL, but
only where Oracle has expressly included in the particular source file's header
the words "Oracle designates this particular file as subject to the "Classpath"
exception as provided by Oracle in the LICENSE file that accompanied this code."

    Linking this library statically or dynamically with other modules is making
    a combined work based on this library.  Thus, the terms and conditions of
    the GNU General Public License cover the whole combination.

    As a special exception, the copyright holders of this library give you
    permission to link this library with independent modules to produce an
    executable, regardless of the license terms of these independent modules,
    and to copy and distribute the resulting executable under terms of your
    choice, provided that you also meet, for each linked independent module,
    the terms and conditions of the license of that module.  An independent
    module is a module which is not derived from or based on this library.  If
    you modify this library, you may extend this exception to your version of
    the library, but you are not obligated to do so.  If you do not wish to do
```

(continues on next page)

(continued from previous page)

```
    so, delete this exception statement from your version.




ADDITIONAL INFORMATION ABOUT LICENSING

Certain files distributed by Oracle America, Inc. and/or its affiliates are
subject to the following clarification and special exception to the GPLv2,
based on the GNU Project exception for its Classpath libraries, known as the
GNU Classpath Exception.

Note that Oracle includes multiple, independent programs in this software
package.  Some of those programs are provided under licenses deemed
incompatible with the GPLv2 by the Free Software Foundation and others.
For example, the package includes programs licensed under the Apache
License, Version 2.0 and may include FreeType. Such programs are licensed
to you under their original licenses.

Oracle facilitates your further distribution of this package by adding the
Classpath Exception to the necessary parts of its GPLv2 code, which permits
you to use that code in combination with other independent modules not
licensed under the GPLv2. However, note that this would not permit you to
commingle code under an incompatible license with Oracle's GPLv2 licensed
code by, for example, cutting and pasting such code into a file also
containing Oracle's GPLv2 licensed code and then distributing the result.

Additionally, if you were to remove the Classpath Exception from any of the
files to which it applies and distribute the result, you would likely be
required to license some or all of the other code in that distribution under
the GPLv2 as well, and since the GPLv2 is incompatible with the license terms
of some items included in the distribution by Oracle, removing the Classpath
Exception could therefore effectively compromise your ability to further
distribute the package.

Failing to distribute notices associated with some files may also create
unexpected legal consequences.

Proceed with caution and we recommend that you obtain the advice of a lawyer
skilled in open source matters before removing the Classpath Exception or
making modifications to this package which may subsequently be redistributed
and/or involve the use of third party software.
```

Chapter 2

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.

Chapter 3

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.