RTI Connext Micro

User's Manual

Version 4.0.1

Your systems.
Working as one.

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

Introduction

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.

1.1.1 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.

1.2 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.

1.2.1 DDS Entity Support

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

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

1.2.2 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

1.3 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.

1.4 OMG DDS Specification

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

1.5 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

Chapter 2

Installation

2.1 Installing the RTI Connext Micro Package

RTI Connext Micro is provided in the following target package:

`rti_connext_dds_micro-4.0.1.rtipkg`

You must first install *RTI Connext Professional* 7.3.0 before installing `rti_connext_dds_micro-4.0.1.rtipkg`.

Once installed, you will see a directory `rti_connext_dds_micro-4.0.1` in the *RTI Connext Professional* installation directory. This installation directory contains this documentation, the *rtiddsgen* code generation tool, and example source code.

Note that a JRE is needed to execute *rtiddsgen*. The default JRE is the JRE installed in the *Connext Drive* or *Connext Professional* installation where *Connext Micro* is installed. If a different JRE is needed, please refer to section *Generating Type Support with rtiddsgen* for more information about *rtiddsgen*.

Note: 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.

2.2 Setting Up Your Environment

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-7.3.0/rti_connext_micro-4.0.1`. If you copied *RTI Connext Micro* to another place, set `RTIMEHOME` to point to that location.

2.3 Building Connext Micro

This section is for users who are already familiar with CMake and may have built earlier versions of *Connext Micro*. The sections following describe the process in detail and are recommended for everyone building *Connext Micro*.

This section assumes that the *Connext Micro* source-bundle has been downloaded and installed and that CMake is available.

1. Make sure CMake is in the path.
2. Run `rtime-make`.

   On UNIX® systems:

   ```
   cd <rti_me install directory>
   # you should see directories like doc/ lib/ rtiddsgen/ src/
   # and CMakeLists.txt

   resource/scripts/rtime-make --target x64Linux4gcc7.3.0 \
               -G "Unix Makefiles" --build
   ```

   On Windows® systems:

   ```
   cd <rti_me install directory>
   # you should see directories like doc/ lib/ rtiddsgen/ src/
   # and CMakeLists.txt

   resource\scripts\rtime-make --target x64Win64VS2017 \
               -G "NMake Makefiles" --build
   ```

   Note: Running the command above will build debug libraries. To build release libraries, add the following flag:

   ```
   --config Release
   ```

3. You will find the *Connext Micro* libraries here:

   On UNIX-based systems:

   ```
   # <rti_me install directory>/lib/x64Linux4gcc7.3.0
   ```

   On Windows systems:

   ```
   # <rti_me install directory>\lib\x64Win64VS2017
   ```

Note: `rtime-make` uses the platform specified with `--target` to determine a few settings needed by *Connext Micro*. Please refer to *Preparing for a Build* for details.

Chapter 3

Building Connext Micro

3.1 RTI Connext Micro Platforms

RTI Connext Micro is a source product and can be ported to all reference platforms that RTI supports; see *Reference Platforms* below. However, RTI does not test and validate the libraries on all permutations of CPU types, compiler version and OS version.

3.1.1 Reference Platforms

RTI provides the platform-dependent layers for porting *Connext Micro* for the following reference platforms:

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

3.1.2 Known Customer Platforms

RTI Connext Micro has been ported to a number of platforms by our customers, such as:

- uC/OS™
- uLinux
- Win32
- Android™
- iOS®
- TI's Stellaris® Arm® Cortex®-M3 and -M4 with only TI device drivers, no OS

- Baremetal - Arm Cortex-M4
- INTEGRITY®-178
- VxWorks 653 2.x, 3.x
- DDC-I Deos™
- LynxOS®-178
- VOS™

RTI Connext Micro is known to run with the following network stacks:

- BSD® socket-based stack
- Windows Socket library
- VxWorks Network stack
- ThreadX Network stack
- RTNet®
- lwIP (event and blocking mode)
- QNX Network stack
- GHS IPFlite and general purpose stack

3.2 Building the Source for Common Platforms

3.2.1 Introduction

RTI Connext Micro has been engineered for reasonable portability to common platforms and environments, such as Darwin, iOS, Linux, and Windows. This section explains how to build the *Connext Micro* source-code. The focus of this section is building *Connext Micro* for an architecture supported by RTI (please refer to *RTI Connext Micro Platforms* for more information). Please refer to *Porting Connext Micro* for documentation on how to port *Connext Micro* to an *unsupported* architecture.

This section is written for developers and engineers with a background in software development. We recommend reading this section in order, as one subsection may refer to or assume knowledge about concepts described in a preceding subsection.

3.2.2 The Host and Target 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 runs *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

RTI 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 *RTI 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).

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

Connext Micro has been designed to run on a wide variety of targets. For example, *Connext Micro* can be ported to run with no OS, an RTOS, GNU libc or a non-standard C library etc. This section only lists the minimum requirements. Please refer to *Porting Connext Micro* for how to port *Connext Micro*.

The target machine must:

- support 8, 16, and 32-bit signed and unsigned integer. Note that a 16 bit CPU (or even 8 bit) is supported as long as the listed types are supported.

  Connext Micro supports 64 bit CPUs, and it does not use any native 64 bit quantities internally.

The target compiler should:

- have a C compiler that is C99 compliant. Note that many non-standard compilers work, but may require additional configuration.
- have a C++ compiler that is C++98 compliant.

The remainder of this manual assumes that the target environment is one supported by RTI:

- POSIX (Linux, Darwin, QNX®, VOS, iOS, Android).
- VxWorks 6.9 or later.
- Windows.
- QNX.

3.2.3 Overview of the Connext Micro Source Bundle

The *Connext Micro* source is available from the RTI support portal. If you do not have access, please contact RTI Support. The source-code is exactly the same as developed and tested by RTI. No filtering or modifications are performed, except for line-ending conversion for the Windows source bundle.

The source-bundle is in a directory called **src/** under your *Connext Micro* installation.

```
RTIMEHOME--+-- CmakeLists.txt
           |
           +-- build -- cmake --+-- Debug --+-- <ARCH> -- <project-files>
           |                    |
           |                    |
           |                    +-- Release --+-- <ARCH> -- <project-files>
           +-- doc --
           |
           +-- example
           |
           +-- include
           |
           +-- lib +-- <ARCH> -- <libraries>
```

(continues on next page)

(continued from previous page)

```
          |
          +-- resource --+-- cmake
          |              |
          |              +-- scripts
          |
          +-- rtiddsgen
          |
          +-- rtiddsmag
          |
          +-- src
```

In this section, `RTIMEHOME` refers to the root directory where RTI archives are extracted and installed.

Directory Structure

The recommended directory structure is described below 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.
- this directory structure mirrors the structure shipped by RTI. (2).

NOTE 1: This applies to builds using CMake. To build in a custom environment, please refer to *Custom Build Environments*.

NOTE 2: The path to an installation of *rtiddsgen*, likely from a bundle shipped by RTI, will also have to be specified separately.

CMakeLists.txt is the main input file to CMake and is used to generate build files.

The *RTIMEHOME/include* directory contains the public header files. By default it is identical to *RTIMEHOME/include*. However, custom ports will typically add files to this directory.

The *RTIMEHOME/src* directory contains the *Connext Micro* source files. RTI does not support modifications to these files unless explicitly stated in the porting guide. A custom port will typically add specific files to this directory.

The *RTIMEHOME/build* directory is empty by default. CMake generates one set of build-files for each configuration. A build configuration can be an architecture, *Connext Micro* options, language selection, etc. This directory will contain CMake generated build-files per architecture per configuration. By convention the *Debug* directory is used to generate build-files for debug libraries and the *Release* directory is used for release libraries.

The *RTIMEHOME/lib* directory is empty by default. All libraries successfully built with the CMake generated build-files, regardless of which generator was used, will be copied to the *RTIMEHOME/lib* directory.

The following naming conventions are used regardless of the build-tool:

- 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.

The following libraries are built:

- *rti_me* - the core library, including the DDS C API
- *rti_me_discdpde* - the Dynamic Participant Dynamic Endpoint (DPDE) plugin
- *rti_me_discdpse* - the Dynamic Participant Static Endpoint (DPSE) plugin
- *rti_me_rhsm* - the Reader History plugin
- *rti_me_whsm* - the Writer History plugin
- *rti_me_netioshmem* - the Shared Memory Transport
- *rti_me_netiosdm* - the Zero Copy over shared memory transport library
- *rti_me_appgen* - the Application Generation plugin
- *rti_me_cpp* - the C++ API

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

1. *rti_me_appgen* (if using the Application Generation plugin)
2. *rti_me_netioshmem* (if using the Shared Memory Transport)
3. *rti_me_netiosdm* (if using the Zero Copy transport)
4. *rti_me_discdpde* (if using DPDE)
5. *rti_me_discdpse* (if using DPSE)
6. *rti_me_rhsm rti_me_whsm rti_me* (always required)

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

1. *rti_me_appgen* (if using the Application Generation plugin)
2. *rti_me_cpp rti_me_netioshmem rti_me_netiosdm rti_me_discdpde rti_me_discdpse rti_me_rhsm rti_me_whsm rti_me* (always required)

Note: The names above are the RTI library names. Depending on the target architecture, the library name is prefixed with *lib* and the library suffix also varies between target architectures, such as .so, .dylib, etc.

For example:

- rti_mezd indicates a static debug library
- rti_me indicates a dynamically linked release library

3.2.4 Compiling Connext Micro

This section describes in detail how to compile *Connext Micro* using CMake. It starts with an example that uses the included scripts followed by a section showing how to build manually.

CMake, available from www.cmake.org, 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 has no dependencies on external tools.

NOTE: It is not required to use CMake. Please refer to *Custom Build Environments* for other ways to build *Connext Micro*.

Building Connext Micro 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.

On UNIX-based systems:

```
RTIMEHOME/resource/scripts/rtime-make --config Debug --target x64Linux4gcc7.3.0 \
                      -G "Unix Makefiles" --build
```

On Windows systems:

```
RTIMEHOME\resource\scripts\rtime-make --config Debug --target x64Win64VS2017 \
                   -G "Visual Studio 15 2017" --build
```

Explanation of arguments:

- `--config Debug` : Create Debug build.
- `--target <target>` : The target for the sources to be built.
- `--build Build`: The generated project files.

On UNIX-based systems:

- If gcc is part of the name, GCC is assumed.
- If clang is part of the name, clang is assumed.

On Windows systems:

- If Win32 is part of the name, a 32 bit Windows build is assumed.
- If Win64 is part of the name, a 64 bit Windows build is assumed.

To get a list of all the options:

```
rtime-make -h
```

To get help for a specific target:

```
rtime-make --target <target> --help
```

Manually Building with CMake

Preparing for a Build

As mentioned, it is recommended to create 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 tool-chain files in *Cross-Compiling Connext Micro* uses the **RTIME_TARGET_NAME** variable to determine various compiler options and selections.

RTI uses the following name format:

```
{cpu}{OS}{compiler}_{config}
```

In order to avoid a naming conflict with RTI, the following name format is recommended:

```
{prefix}_{cpu}{OS}{compiler}_{config}
```

Some examples:

- acme_ppc604FreeRTOSgcc4.6.1 - *Connext Micro* for a PPC 604 CPU running FreeRTOS compiled with gcc 4.6.1, compiled by acme.
- acme_i86Win32VS2015 - *Connext Micro* for an i386 CPU running Windows XP or higher compiled with Visual Studio 2015, compiled by acme.
- acme_i86Linux4gcc4.4.5_test - a test configuration build of *Connext Micro* for an i386 CPU running Linux 3 or higher compiled with gcc 4.4.5, compiled by acme.

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 for Connext Micro Using the CMake GUI

Start the CMake GUI, either from a terminal window or a menu.

Please note that the Cmake GUI does *not* set the **CMAKE_BUILD_TYPE** variable. This variable is used to determine the names of the *Connext Micro* libraries. Thus, it is necessary to add **CMAKE_BUILD_TYPE** manually and specify either Debug or Release. To add this variable manually, click the 'Add Entry' button, specify the name as a string type.

As an alternative, rtime-make's `--gui` option can be used. This option starts the CMake and also adds the **CMAKE_BUILD_TYPE** option when the CMake GUI exits.

Please note that when using Visual Studio or Xcode, it is important to build the same configuration as was specified with rtime-make's `--config` option. While it is possible to build a different configuration from the IDE, selecting a different configuration does *not* update the build configuration generated for *Connext Micro*.

The GUI should be started from the `RTIMEHOME` directory. If this is not the case, check that:

- The source directory is the location of `RTIMEHOME`.
- The binary directory is the location of `RTIMEBUILD`.

With the CMake GUI running:

- Press 'Configure'.
- Select a generator. You must have a compatible tool installed to process the generated files.
- Select 'Use default native compilers'.
- Press 'Done'.
- Press 'Configure'.
- Check 'Grouped'.
- Expand RTIME and select your build options. All available build options for *Connext Micro* are listed here.
- Type a target name for **RTIME_TARGET_NAME**. This should be the same as the *<name>* used to create the `RTIMEBUILD` directory, that is the `RTIMEBUILD` should be on the form *<path>/<RTIME_TARGET_NAME>*.
- Press 'Configure'. All red lines should disappear. Due to how CMake works, it is strongly recommended to always press 'Configure' whenever a value is changed for a variable. Other variables may depend on the modified variable and pressing 'Configure' will mark those with a red line. No red lines means everything has been configured.
- Press 'Generate'. This creates the build-files in the `RTIMEBUILD` directory. Whenever an option is changed and Configure is re-run, press Generate again.
- Exit the GUI.

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) change to the RTIMEBUILD directory and 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, when using Xcode or 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
```

Creating Build Files for Connext Micro Using CMake 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 using the following arguments:

```
cmake -G <generator> -DCMAKE_BUILD_TYPE=<Debug | Release> \
      -DCMAKE_TOOLCHAIN_FILE=<toolchain file>  \
      -DRTIME_TARGET_NAME=<target-name>
```

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
```

CMake Flags used by Connext Micro

The following CMake flags (-D) are understood by *Connext Micro* and may be useful when building outside of the source bundle installed by RTI. An example would be incorporating the *Connext Micro* source in a project tree and invoking cmake directly on the CMakeLists.txt provided by *Connext Micro*.

- `-DRTIME_TARGET_NAME=\<name\>` - The name of the target (equivalant to `--target` to rtime-make). The default value is the name of the source directory.
- `-DRTIME_CMAKE_ROOT=\<path\>` - Where to place the CMake build files. The default value is *<source>/build/cmake*.
- `-DRTIME_BUILD_ROOT=\<path\>` - Where to place the intermediate build files. The default value is *<source>/build*.
- `-DRTIME_SYSTEM_FILE=\<file\>` or an empty string - This file can be used to set the PLATFORM_LIBS variable used by *Connext Micro* to link with. If an empty string is specified no system file is loaded. This option may be useful when cmake can detect all that is needed. The default value is not defined, which means try to detect the system to build for.
- `-DRTI_NO_SHARED_LIB=true` - Do not build shared libraries. The default is undefined, which means shared libraries are built. NOTE: This flag must be undefined to build shared libraries. Setting the value to false is not supported.
- `-DRTI_MANUAL_BUILDID=true` - Do not automatically generate a build ID. The default value is undefined, which means generate a new build each time the libraries are built. Setting the value to false is not supported. The build ID is in its own source and only forces a recompile of a few files. Note that it is necessary to generate a build ID at least once (this is done automatically). Also, a build ID is not supported for cmake versions less than 2.8.11 because the TIMESTAMP function does not exist.
- `-DRTIME_DDS_DISABLE_PARTICIPANT_MESSAGE_DATA=false` Disables P2P Message Data inter-participant channel. This channel is needed to use **DDS_AUTOMATIC_LIVELINESS_QOS** and **DDS_MANUAL_BY_PARTICIPANT_LIVELINESS_QOS** with a finite lease duration.

3.2.5 Connext Micro Compile Options

The *Connext Micro* source supports compile-time options. These options are in general used to control:

- Enabling/Disabling features.
- Inclusion/Exclusion of debug information.
- Inclusion/Exclusion of APIs.
- Target platform definitions.

- Target compiler definitions.

NOTE: It is no longer possible to build a single library using CMake. Please refer to *Custom Build Environments* for information on customized builds.

Connext Micro Debug Information

Please note that *Connext Micro* debug information is independent of a debug build as defined by a compiler. In the context of *Connext Micro*, debug information refers to inclusion of:

- Logging of error-codes.
- Tracing of events.
- Precondition checks (argument checking for API functions).

Unless explicitly included/excluded, the following rule is used:

- For CMAKE_BUILD_TYPE = Release, the NDEBUG preprocessor directive is defined. Defining NDEBUG includes logging, but excludes tracing and precondition checks.
- For CMAKE_BUILD_TYPE = Debug, the NDEBUG preprocessor directive is undefined. With NDEBUG undefined, logging, tracing and precondition checks are included.

To manually determine the level of debug information, the following options are available:

- **OSAPI_ENABLE_LOG** (Include/Exclude/Default)
  - Include - Include logging.
  - Exclude - Exclude logging.
  - Default - Include logging based on the default rule.
- **OSAPI_ENABLE_TRACE** (Include/Exclude/Default)
  - Include - Include tracing.
  - Exclude - Exclude tracing.
  - Default - Include tracing based on the default rule.
- **OSAPI_ENABLE_PRECONDITION** (Include/Exclude/Default)
  - Include - Include tracing.
  - Exclude - Exclude tracing.
  - Default - Include precondition checks based on the default rule.

Connext Micro Platform Selection

The *Connext Micro* build system looks for target platform files in *RTIMEHOME/include/osapi*. All files that match *osapi_os_*.h are listed under **RTIME_OSAPI_PLATFORM**. Thus, if a new port is added it will automatically be listed and available for selection.

The default behavior, <auto detect>, is to try to determine the target platform based on header-files. The following target platforms are known to work:

- Linux (posix)
- VOS (posix)
- QNX (posix)
- Darwin (posix)
- iOS (posix)
- Android (posix)
- Win32 (windows)
- VxWorks 6.9 and later (vxworks)

However, for custom ports this may not work. Instead the appropriate platform definition file can be selected here.

Connext Micro Compiler Selection

The *Connext Micro* build system looks for target compiler files in *RTIMEHOME/include/osapi*. All files that match *osapi_cc_*.h are listed under **RTIME_OSAPI_COMPILER**. Thus, if a new compiler definition file is added it will automatically be listed and available for selection.

The default behavior, <auto detect>, is to try to determine the target compiler based on header-files. The following target compilers are known to work:

- GCC (stdc)
- clang (stdc)
- MSVC (stdc)

However, for others compilers this this may not work. Instead the appropriate compiler definition file can be selected here.

Connext Micro UDP Options

Checking the **RTIME_UDP_ENABLE_IPALIASES** disables filtering out IP aliases. Note that this currently only works on platforms where each IP alias has its own interface name, such as eth0:1, eth1:2, etc.

Checking the **RTIME_UDP_ENABLE_TRANSFORMS_DOC** enables UDP transformations in the UDP transport.

Checking the **RTIME_UDP_EXCLUDE_BUILTIN** excludes the UDP transport from being built.

3.2.6 Cross-Compiling Connext Micro

Cross-compiling the *Connext Micro* source-code uses the exact same process described in *Compiling Connext Micro*, but requires a additonal *tool-chain file*. A tool-chain file is a CMake file that describes the compiler, linker, etc. needed to build the source for the target. The *Connext Micro* source bundle includes a few basic, generic tool-chain files for cross-compilation. In general it is expected that users will provide their own cross-compilation tool-chain files.

To see a list of available targets, use `--list` :

```
rtime-make --list
```

By convention, RTI only provides generic tool-chain files that can be used to build for a broad range of targets. For example, the Linux target can be used to build for any Linux architecture as long as it is a self-hosted build. The same is true for Windows and Darwin systems. The VxWorks tool-chain file uses the Wind River environment variables to select the compiler.

For example, to build on a Linux machine with Kernel 2.6 and gcc 4.7.3:

```
rtime-make --target x64Linux4gcc7.3.0 --config Debug --build
```

By convention, a specific name such as i86Linux2.6gcc4.4.5 is expected to only build for a specific target architecture. Note however that this cannot be enforced by the script provided by RTI. To create a target specific tool-chain file, copy the closest matching file and add it to the *RTIMEHOME/source/Unix/resource/CMake/architectures* or *RTIMEHOME/source/windows/resource/CMake/architectures* directory.

Once a tool-chain file has been created, or a suitable file has been found, edit it as needed. Then invoke rtime-make, specifying the new tool-chain file as the target architecture. For example:

```
rtime-make --target i86Linux2.6gcc4.4.5 --config Debug --build
```

3.2.7 Custom Build Environments

The preferred method to build *Connext Micro* is to use CMake. However, in some cases it may be more convenient, or even necessary, to use a custom build environment. For example:

- Embedded systems often have numerous compiler, linker and board specific options that are easier to manage in a managed build.
- The compiler cannot be invoked outside of the build environment, it may be an integral part of the development environment.
- Sometimes better optimization may be achieved if all the components of a project are built together.
- It is easier to port *Connext Micro*.

Importing the Connext Micro Code

The process for importing the *Connext Micro* Source Code into a project varies depending on the development environment. However, in general the following steps are needed:

- Create a new project or open an existing project.
- Import the entire *Connext Micro* source tree from the file-system. Note that some environments let you choose whether to make a copy only link to the original files.
- Add the following include paths:
  - <root>/include
  - <root>/src/dds_c/domain
  - <root>/src/dds_c/infrastructure
  - <root>/src/dds_c/publication
  - <root>/src/dds_c/subscription
  - <root>/src/dds_c/topic
  - <root>/src/dds_c/type
- Add a compile-time definition `-DRTIME_TARGET_NAME="target name"` (note that the ” must be included).
- Add a compile-time definition `-DNDEBUG` for a release build.
- Add a compile-time definition of either `-DRTI_ENDIAN_LITTLE` for a little-endian platform or `-DRTI_ENDIAN_BIG` for a big-endian platform.
- If custom OSAPI definitions are used, add a compile-time definition `-DOSAPI_OS_DEF_H="my_os_file"`.
- If custom compiler definitions are used, add a compile-time definition `-DOSAPI_CC_DEF_H="my_cc_file.h"` .

3.3 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*

3.3.1 Compiling with compatibility for Connext Cert

To compile *Connext Micro* with the Cert-compatibility profile, you must use one of the available CERT architectures. To get a list of available CERT architectures, please use the following command:

```
cd <rti_me install directory>
resource/scripts/rtime-make --list
```

Architectures ending in CERT (e.g, 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`.

3.3.2 Compiling Applications with compatibility for Connext Cert

To compile an application with compatibility for Connext Cert, the application must be compiled with the `RTI_CERT=1` preprocessor flag. This can be achieved with one of the following methods:

- If a CMakeLists.txt file generated with rtiddsgen is used, pass `-DRTIME_CERT=true` to either `rtime-make` or cmake.
- Pass `-DRTI_CERT=1` directly to the C preprocessor

With `rtime-make` on Linux or macOS:

```
resource/scripts/rtime-make -target 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 *Example Generation* for more information about generating examples.

Chapter 4

Getting Started

4.1 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*.

4.2 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*.

4.3 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*).

4.3.1 Registry Configuration

The DomainParticipantFactory, in addition to its standard role of creating and deleting *DomainParticipants*, 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 */
```

(continues on next page)

(continued from previous page)

```
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.

4.4 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))
{
```

(continues on next page)

(continued from previous page)

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

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

4.5 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;
```

(continues on next page)

(continued from previous page)

```
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");

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

4.5.1 Register Type

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

4.5.2 Create Topic of Registered Type

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

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

```
DDS_Topic *topic = NULL;

topic = DDS_DomainParticipant_create_topic(participant,
                                           "Example HelloWorld",
```

(continues on next page)

(continued from previous page)

```
                                             "HelloWorld",
                                             &DDS_TOPIC_QOS_DEFAULT,
                                             NULL,
                                             DDS_STATUS_MASK_NONE);

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

4.5.3 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.

4.6 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.

4.7 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");
    }
}
```

4.7.1 DPSE Discovery: Assert Remote Subscription

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

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

```
struct DDS_SubscriptionBuiltinTopicData rem_subscription_data =
    DDS_SubscriptionBuiltinTopicData_INITIALIZER;

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

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

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 */
}
```

4.7.2 Writing Samples

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

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

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

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

(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.

4.8 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.

4.9 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;
```

(continues on next page)

(continued from previous page)

```
dr_qos.history.depth = 1;
dr_qos.durability.kind = DDS_VOLATILE_DURABILITY_QOS;
dr_qos.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

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

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

4.9.1 DPSE Discovery: Assert Remote Publication

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

```
struct DDS_PublicationBuiltinTopicData rem_publication_data =
    DDS_PublicationBuiltinTopicData_INITIALIZER;

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

(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.

4.9.2 Receiving Samples

Accessing received samples can be done in a few ways:

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

4.9.3 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,
```

(continues on next page)

(continued from previous page)

```
    const struct DDS_SampleInfo *const sample_info,
    DDS_Boolean *dropped)
{
    HelloWorld *hw_sample = (HelloWorld *)sample;

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

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

    return DDS_BOOLEAN_TRUE;
}

...

dr_listener.on_before_sample_commit =
    HelloWorldSubscriber_on_before_sample_commit;
```

For more information, see the *Receiving Data* section in the User's Manual.

4.10 Examples

Connext Micro provides buildable example applications, in the **example/** directory. Each example comes with instructions on how to build and run an application.

- **HelloWorld_transformations**: A HelloWorld example that uses UDP transformations to send encrypted packets using OpenSSL.

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 *Example Generation*.

Note: By default, all the examples link against release libraries. To build release libraries:

```
./resource/scripts/rtime-make --target x64Darwin20clang12.0 --build --config Release
```

To build the examples against the debug libraries, specify the DEBUG option:

```
make DEBUG=Y
```

4.11 Example Generation

The RTI Code Generator available with *Connext Micro* can generate DDS example applications with a type definition file as input.

Note that before running *rtiddsgen*, you might need to add <*Connext Micro* install folder>/rtiddsgen/scripts to your path environment variable folder.

To generate an example:

```
rtiddsgen -example -language <C|C++> [-namespace] <file with type definition>
```

This command 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 be compiled using CMake <https://cmake.org/>_ and the CMakelists.txt file generated by the RTI Code Generator. A README.txt file is also generated with a description of the example and instructions for how to compile and run the examples.

The RTI Code Generator can also generate examples using custom templates by using the option `-exampleTemplate <templateName>`.

To generate an example using a custom template instead of the default one:

```
rtiddsgen -example -exampleTemplate <template name> -language <C|C++> [-namespace] <file␣
↪with type definition>
```

To see the list of the available templates, use the following command:

```
rtiddsgen -showTemplates
```

The output from the command will look similar to this:

```
List of example templates per language:
    - C:
        - cert
        - dpse
        - mag/dpde
        - mag/dpse
        - mag/shared_memory
        - mag/static_udp
        - secure
        - shared_memory
        - static_udp
        - waitsets
    - C++:
```

(continues on next page)

(continued from previous page)

```
        - dpse
        - mag/dpde
        - secure
        - waitsets
    - C++ Namespace:
        - 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>
```

4.11.1 Description of Examples

All examples consist of a publication and subscription pair to send and receive the type provided by user. Two applications are compiled: one to send samples and another to receive samples.

- **Default template** Discovery of endpoints is done with the dynamic-endpoint discovery. 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** The only difference from the default template is that the discovery of endpoints is done with static-endpoint discovery. Static-endpoint discovery uses function calls to statically assert information about remote endpoints belonging to remote DomainParticipants.
- **secure** The only difference from the default template is that this example uses secure communication.
- **shared_memory** The only difference from the default template is 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** The only difference from the default template is 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).
- **mag** All the example templates whose names start with 'mag' use the Micro Application Generator (MAG) to create the DDS entities. This means that the DDS entities and

their configuration are generated from an XML file, instead of by calling DDS APIs in the application.

- **mag/dpde** The same as the default template, but DDS entities are created by using MAG instead of by calling by calling DDS APIs in the application.
- **mag/dpse** The same as the 'dpse' template, but DDS entities are created by using MAG instead of by calling DDS APIs in the application.
- **mag/shared_memory** The same as the 'shared_memory' template, but DDS entities are created by using MAG instead of by calling DDS APIs in the application.
- **mag/static_udp** The same as the 'static_udp' template, but DDS entities are created by using MAG instead of by calling DDS APIs in the application.

4.11.2 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*.

For a 'mag' example, you can find the setting in the file '<File name used to generate example>Qos.xml' in the XML element <allow_interfaces_list>. For other (non-'mag') examples, use the option **`-udp_interface <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 x64Linux4gcc7.3.0 --source-dir . \
     -G "Unix Makefiles" --delete [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] \
     [-DRTIME_MAG_FILES=<XML file>] [-DRTIME_LINK_SHMEM_LIBS=true]
```

macOS

```
cd <directory with generated example>

rtime-make --config <Debug|Release> --build --target x64Darwin20clang12.0 --source-dir .␣
↪\
     -G "Unix Makefiles" --delete [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] \
     [-DRTIME_MAG_FILES=<XML file>] [-DRTIME_LINK_SHMEM_LIBS=true]
```

Windows

```
cd <directory with generated example>

```

(continues on next page)

(continued from previous page)

```
rtime-make.bat --config <Debug|Release> --build --target x64Win64VS2017 --source-dir . \
     -G "Visual Studio 15 2017" --delete [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE_eq_
↪true] \
     [-DRTIME_MAG_FILES_eq_<XML file>] [-DRTIME_LINK_SHMEM_LIBS_eq_true]
```

Warning: RTI recommends using the same toolchain file to compile generated examples as was used to compile *Connext Micro* (as described in *Building Connext Micro* and *Building the Source for Common Platforms*).

For example, if *Connext Micro* was compiled with `--target x64Linux4gcc7.3.0`, then the example applications should also be compiled with `--target x64Linux4gcc7.3.0`. 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_MAG_FILES=<XML file>]␣
↪[-DRTIME_LINK_SHMEM_LIBS=true] \
      [-DCMAKE_BUILD_TYPE=<Debug|Release>] -G "Unix Makefiles" -B./<your build directory>
↪ -H. -DRTIME_TARGET_NAME=x64Linux4gcc7.3.0

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

macOS

```
cmake [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] [-DRTIME_MAG_FILES=<XML file>]␣
↪[-DRTIME_LINK_SHMEM_LIBS=true] \
      [-DCMAKE_BUILD_TYPE=<Debug|Release>] -G "Unix Makefiles" -B./<your build directory>
↪ -H. -DRTIME_TARGET_NAME=x64Darwin20clang12.0

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

Windows

```
cmake [-DRTIME_IDL_ADD_REGENERATE_TYPESUPPORT_RULE=true] [-DRTIME_MAG_FILES=<XML file>]␣
↪[-DRTIME_LINK_SHMEM_LIBS=true] \
      [-DCMAKE_BUILD_TYPE=<Debug|Release>] -G "Visual Studio 15 2017" -B./<your build␣
↪directory> -H. -DRTIME_TARGET_NAME=x64Win64VS2017

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_MAG_FILES=<XML file with Application Generation definitions>` adds a rule to generate application support files from XML.
- `-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'.

4.11.3 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.

For a 'mag' example, you can find the setting in the file `<File name used to generate example>Qos.xml` in the XML element <allow_interfaces_list>. For other (non-'mag') examples, use the option `-udp_interface <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/x64Linux3gcc4.8.2/<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/x64Linux3gcc4.8.2/<Type definition file name>_publisher [-domain <Domain_ID> -peer
↪<address>] \
            [-sleep <sleep_time>] [-count <seconds_to_run>] [-udp_intf <interface name>]
```

Note that shared memory examples only accept the following options: `[-domain <Domain_ID>]`, `[-sleep <sleep_time>]` and `[-count <seconds_to_run>]`.

MAG examples only accept the following options: `[-sleep <sleep_time>]` and `[-count <seconds_to_run>]`.

Chapter 5

User's Manual

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).

5.1.1 Introduction to the Type System

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

Your application can have any number of user data types. 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

5.1.2 Creating User Data Types with IDL

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

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

5.1.3 Working with DDS Data Samples

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

In C:

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

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

In Traditional C++:

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

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

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

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.

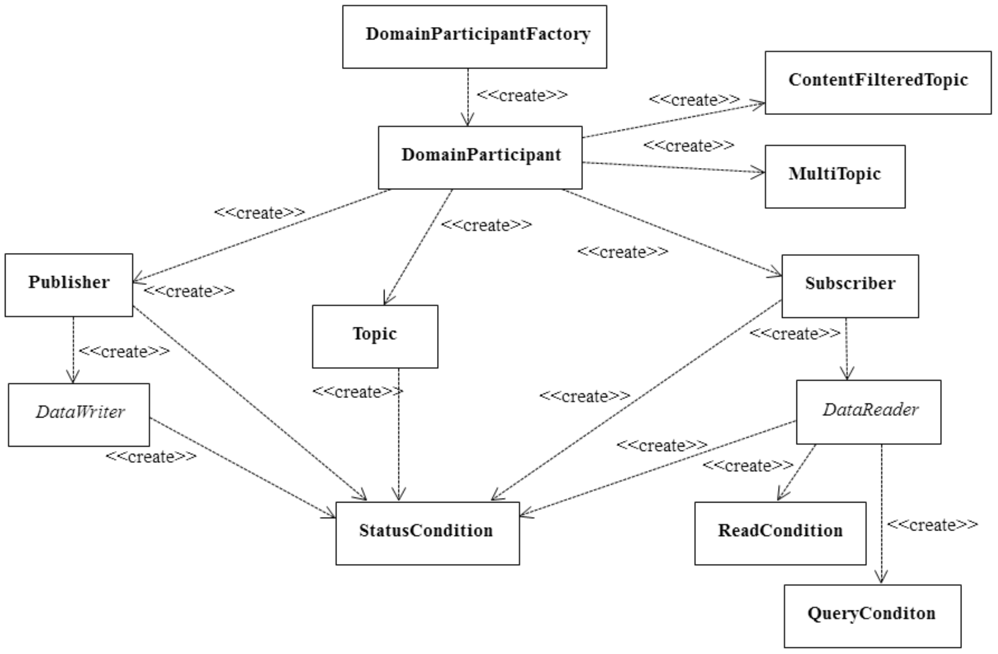


Figure 5.1: Overview of DDS Entities

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.

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.

5.3.1 Preview: Steps to Sending Data

To send DDS samples of a data instance:

1. Create and configure the required *Entities*:

   a. Create a *DomainParticipant*.

   b. Register user data types with the *DomainParticipant*. For example, the '**FooDataType**'.

   c. Use the *DomainParticipant* to create a *Topic* with the registered data type.

   d. Use the *DomainParticipant* to create a *Publisher*.

   e. Use the *Publisher* or *DomainParticipant* to create a *DataWriter* for the *Topic*.

   f. Use a type-safe method to cast the generic *DataWriter* created by the *Publisher* to a type-specific *DataWriter*. For example, '**FooDataWriter**'. Optionally, register data instances with the *DataWriter*. If the *Topic*'s user data type contain key fields, then registering a data instance (data with a specific key value) will improve performance when repeatedly sending data with the same key. You may register many different data instances; each registration will return an instance handle corresponding to the specific key value. For non-keyed data types, instance registration has no effect.

2. Every time there is changed data to be published:

   a. Store the data in a variable of the correct data type (for instance, variable '**Foo**' of the type '**FooDataType**').

   b. Call the **FooDataWriter**'s **write()** operation, passing it a reference to the variable '**Foo**'.

      - For non-keyed data types or for non-registered instances, also pass in **DDS_HANDLE_NIL**.
      - For keyed data types, pass in the instance handle corresponding to the instance stored in 'Foo', if you have registered the instance previously. This means that the data stored in 'Foo' has the same key value that was used to create instance handle.

   c. The **write()** function will take a snapshot of the contents of '**Foo**' and store it in *Connext 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.

5.3.2 Publishers

An application that intends to publish information needs the following *Entities*: *DomainParticipant*, *Topic*, *Publisher*, and *DataWriter*. All *Entities* have a corresponding specialized *Listener* and a set of 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*.

5.3.3 DataWriters

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

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

For more details on creating, deleting, and setting up *DataWriters*, see replace:: the DataWriters section in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

5.3.4 Publisher/Subscriber QosPolicies

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

5.3.5 DataWriter QosPolicies

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

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.

5.4.1 Preview: Steps to Receiving Data

There are three ways to receive data:

- Your application can explicitly check for new data by calling a *DataReader's* **read()** or **take()** operation. This method is also known as *polling for data*.
- Your application can be notified 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*.

5.4.2 Subscribers

An application that intends to subscribe to information needs the following *Entities*: *DomainParticipant*, *Topic*, *Subscriber*, and *DataReader*. All *Entities* have a corresponding specialized *Listener* and a set of 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).

5.4.3 DataReaders

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

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

5.4.4 Using DataReaders to Access Data (Read & Take)

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

5.4.5 Subscriber QosPolicies

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

5.4.6 DataReader QosPolicies

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

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.

5.5.1 Fundamentals of DDS Domains and DomainParticipants

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

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

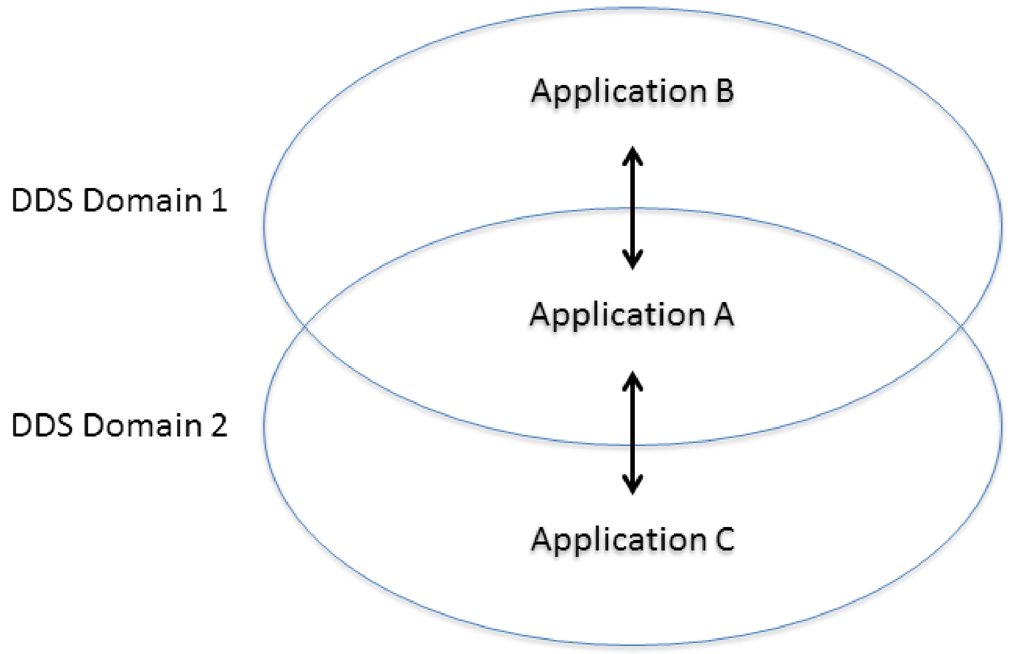


Figure 5.2: Relationship between Applications and DDS Domains
Applications can belong to multiple DDS domains—*A* belongs to DDS domains 1 and 2. Applications in the same DDS domain can communicate with each other, such as *A* and *B*, or *A* and *C*. Applications in different DDS domains, such as *B* and *C*, are not 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*).

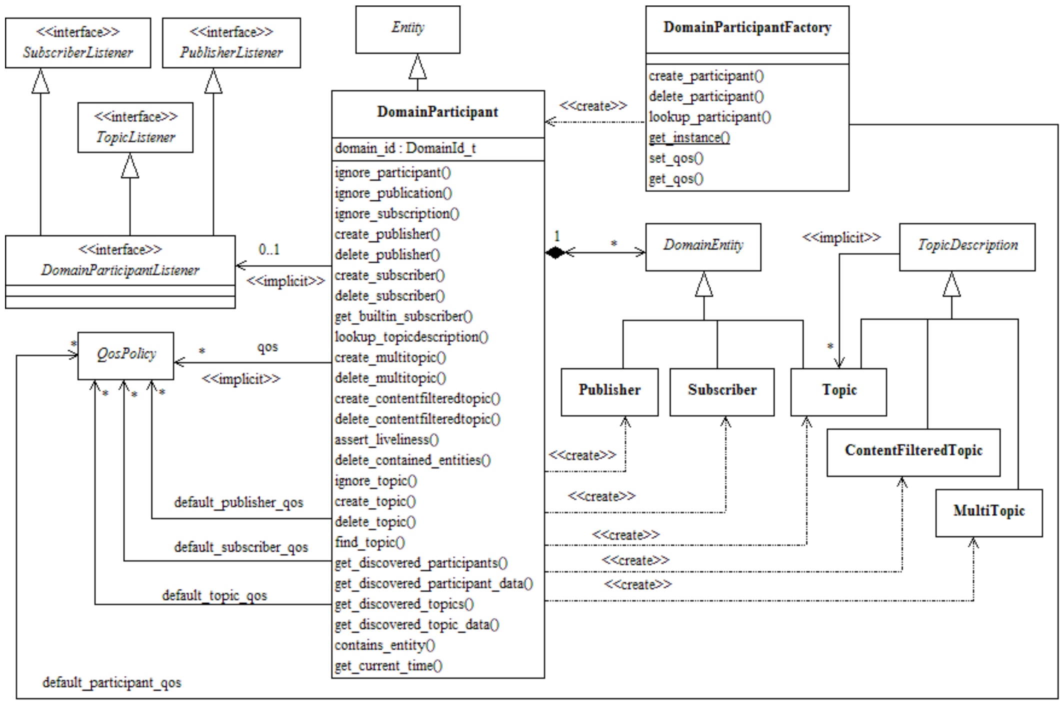


Figure 5.3: DDS Domain Module
Note: MultiTopics are not supported.

5.5.2 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.

5.6 Transports

5.6.1 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.

5.6.2 Transport Registration

RTI Connext Micro supports different transports and transports must be registered with *RTI Connext Micro* before they can be used. A transport must be given a name when it is registered and this name is later used when configuring discovery and user-traffic. A transport name cannot exceed 7 UTF-8 characters.

The following example registers the UDP transport with *RTI Connext Micro* and makes it available to all DDS applications within the same memory space. Please note that each DDS applications creates its *own* instance of a transport. Resources are *not* shared between instances of a transport unless stated.

For example, to register two UDP transports with the names myudp1 and myudp2, the following code is required:

```
DDS_DomainParticipantFactory *factory;
RT_Registry_T *registry;
struct UDP_InterfaceFactoryProperty udp_property;

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

/* Set UDP properties */
if (!RT_Registry_register(registry,"myudp1",
                          UDP_InterfaceFactory_get_interface(),
                          &udp_property._parent._parent,NULL))
{
    return error;
}

/* Set UDP properties */
if (!RT_Registry_register(registry,"myudp2",
                          UDP_InterfaceFactory_get_interface(),
                          &udp_property._parent._parent,NULL))
{
    return error;
}
```

Before a DomainParticipant can make use of a registered transport, it must enable it for use within the DomainParticipant. This is done by setting the TransportQoS. For example, to enable only myudp1, the following code is required (error checking is not shown for clarity):

```
DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,1);
DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,1);
*DDS_StringSeq_get_reference(&dp_qos.transports.enabled_transports,0) =
                                                 REDA_String_dup("myudp1");
```

To enable both transports:

```
DDS_StringSeq_set_maximum(&dp_qos.transports.enabled_transports,2);
DDS_StringSeq_set_length(&dp_qos.transports.enabled_transports,2);
*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.

5.6.3 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.

5.6.4 Transport Port Number

The port number of a locator is not directly configurable. Rather, it is configured indirectly by the DDS_WireProtocolQosPolicy (rtps_well_known_ports) of the *DomainParticipant's* QoS, where a well-known, interoperable RTPS port number is assigned.

5.6.5 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.

5.6.6 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.

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;
```

(continues on next page)

(continued from previous page)

```
    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,
            "_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 */
```

(continues on next page)

(continued from previous page)

```
    }
}
```

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 = ...;
```

(continues on next page)

(continued from previous page)

```
/* The priority is platform dependent, it is passed directly to the OS */
shmem_property.recv_thread_property.priority = ...;

if (!RT_Registry_register(registry, "_shmem",
                          SHMEM_InterfaceFactory_get_interface(),
                          (struct RT_ComponentFactoryProperty*)&shmem_property,
                          NULL))
{
    /* ERROR */
}
```

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.

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.

5.6.7 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;
```

(continues on next page)

(continued from previous page)

```
    /* 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
     */
    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

All the configuration of the UDP transport is done via the UDP_InterfaceFactoryProperty.

```
struct UDP_InterfaceFactoryProperty
{
    /* Inherited from */
    struct NETIO_InterfaceFactoryProperty _parent;

    /* Sequence of allowed interface names */
    struct REDA_StringSeq allow_interface;

    /* Sequence of denied interface names */
    struct REDA_StringSeq deny_interface;

    /* The size of the send socket buffer */
    RTI_INT32 max_send_buffer_size;

    /* The size of the receive socket buffer */
    RTI_INT32 max_receive_buffer_size;

    /* The maximum size of the message which can be received */
    RTI_INT32 max_message_size;

    /* The maximum TTL */
    RTI_INT32 multicast_ttl;

#ifndef RTI_CERT
    struct UDP_NatEntrySeq nat;
#endif

    /* The interface table if interfaces are added manually */
    struct UDP_InterfaceTableEntrySeq if_table;

    /* The network interface to use to send to multicast */
    REDA_String_T multicast_interface;

    /* If this should be considered the default UDP interfaces if
     * no other UDP interface is found to handle a route
     */
    RTI_BOOL is_default_interface;

    /* Disable reading of available network interfaces using system
     * information and instead rely on the manually configured
     * interface table
     */
    RTI_BOOL disable_auto_interface_config;

    /* Thread properties for each receive thread created by this
     * NETIO interface.
     */
    struct OSAPI_ThreadProperty recv_thread;
```

(continues on next page)

(continued from previous page)

```
    /* Bind to specific interfaces
         */
    RTI_BOOL enable_interface_bind;

    struct UDP_TransformRuleSeq source_rules;

    /* Rules for how to transform sent UDP payloads based on the
     * destination address.
     */
    struct UDP_TransformRuleSeq destination_rules;

    /* Determines how regular UDP is supported when transformations
     * are supported.
     */
    UDP_TransformUdpMode_T transform_udp_mode;

    /* The locator to use for locators that have transformations.
     */
    RTI_INT32 transform_locator_kind;
};
```

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)->
```

(continues on next page)

(continued from previous page)

```
                            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
 * 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.

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.

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
};

struct RT_ComponentFactoryI*
MyUdpTransformFactory_get_interface(void)
{
    return &MyUdpTransformFactory_fv_Intf;
}

/*ce \brief Method to register this transformation in a registry
 */
RTI_BOOL
MyUdpTransformFactory_register(RT_Registry_T *registry,
                               const char *const name,
                               struct MyUdpTransformFactoryProperty *property)
{
    return RT_Registry_register(registry, name,
                        MyUdpTransformFactory_get_interface(),
                        &property->_parent, NULL);
}

/*ce \brief Method to unregister this transformation from a registry
 */
RTI_BOOL
MyUdpTransformFactory_unregister(RT_Registry_T *registry,
            const char *const name,
            struct MyUdpTransformFactoryProperty **property)
{
    return RT_Registry_unregister(registry, name,
                            (struct RT_ComponentFactoryProperty**)property,
```

(continues on next page)

(continued from previous page)

```
                              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:
       OSAPILog_set_verbosity(OSAPI_LOG_VERBOSITY_WARNING);
```

(continues on next page)

(continued from previous page)

```
     */
    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(),
                              &discovery_plugin_properties._parent,
```

(continues on next page)

(continued from previous page)

```
                              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)
{
    DDS_Entity *entity;
```

(continues on next page)

(continued from previous page)

```
    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;
    }
    if (!RT_Registry_unregister(registry, DDSHST_READER_DEFAULT_HISTORY_NAME, NULL,␣
↪NULL))
```

(continues on next page)

(continued from previous page)

```
    {
        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.

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.

5.7.1 What is Discovery?

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

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

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

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

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

Simple Participant Discovery

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

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

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

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

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

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

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

Simple Endpoint Discovery

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

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

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

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

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

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

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

5.7.2 Configuring Participant Discovery Peers

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

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

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.

5.7.3 Configuring Initial Peers and Adding Peers

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

DiscoveryQosPolicy_initial_peers is an empty sequence by default, 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.

5.7.4 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.

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*.

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;

/* 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)
{
```

(continues on next page)

(continued from previous page)

```
    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*.

5.8 User Discovery Data

5.8.1 Introduction

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

There are three User Discovery Data QoS policies:

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

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

5.8.2 Resource Limits

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

- `participant_user_data_max_length`
- `topic_data_max_length`

- `publisher_group_data_max_length`
- `subscriber_group_data_max_length`
- `writer_user_data_max_length`
- `reader_user_data_max_length`

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

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

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

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

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

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

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

5.8.3 Propagating User Discovery Data

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

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

```
struct DDS_SubscriptionBuiltinTopicData rem_subscription_data =
   DDS_SubscriptionBuiltinTopicData_INITIALIZER;

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

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

rem_subscription_data.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;

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

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

5.8.4 Accessing User Discovery Data

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

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

- DDS_DomainParticipant_get_discovered_participants
- DDS_DomainParticipant_get_discovered_participant_data

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

- DDS_DataWriter_get_matched_subscriptions
- DDS_DataWriter_get_matched_subscription_data
- DDS_DataReader_get_matched_publications
- DDS_DataReader_get_matched_publication_data

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

For example, to retrieve information on discovered participants:

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

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

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

/* 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 */
```

(continues on next page)

(continued from previous page)

```
    }

    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 */
    }
}
```

5.8.5 QoS Policies

USER_DATA

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

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

The structure for the USER_DATA QosPolicy includes just one field, as seen in Table 5.1. The field is a sequence of octets that translates to a contiguous buffer of bytes whose contents and length are set by the user. The maximum size for the data is set in the DomainParticipantResourceLimitsQosPolicy. (See *Resource Limits*.)

Table 5.1: DDS_UserDataQosPolicy

Type	Field Name	Description
DDS_OctetSeq	value	Empty by default

TOPIC_DATA

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

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

The structure for the TOPIC_DATA QosPolicy includes just one field, as seen in Table 5.2. The field is a sequence of octets that translates to a contiguous buffer of bytes whose contents and length are set by the user. The maximum size for the data is set in the DomainParticipantResourceLimitsQosPolicy. (See *Resource Limits*)

Table 5.2: DDS_TopicDataQosPolicy

Type	Field Name	Description
DDS_OctetSeq	value	Empty by default

GROUP_DATA

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

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

The structure for the TOPIC_DATA QosPolicy includes just one field, as seen in Table 5.3. 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 5.3: DDS_GroupDataQosPolicy

Type	Field Name	Description
DDS_OctetSeq	value	Empty by default

5.9 Partitions

5.9.1 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 5.4 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.

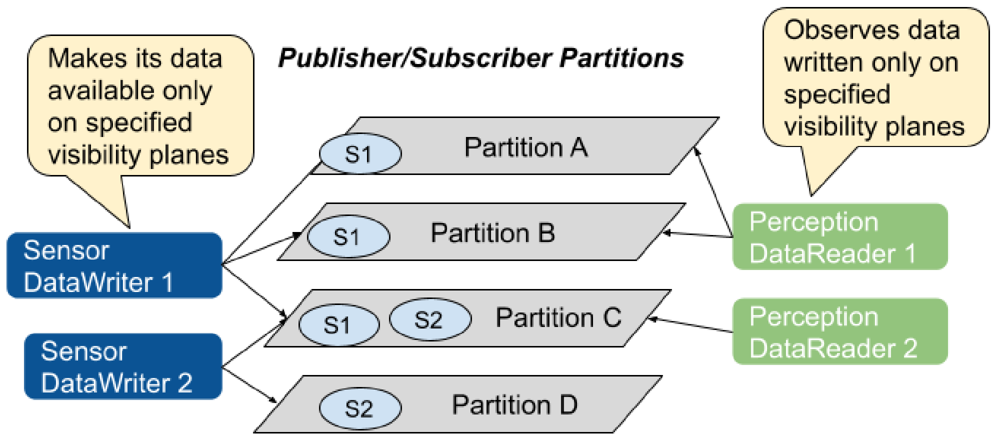

Makes its data available only on specified visibility planes
Publisher/Subscriber Partitions
Observes data written only on specified visibility planes
S1
Partition A
S1
Partition B
S1
S2
Partition C
S2
Partition D
Sensor DataWriter 1
Sensor DataWriter 2
Perception DataReader 1
Perception DataReader 2

Figure 5.4: 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.

5.9.2 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.

5.9.3 Pattern matching for PARTITION names

You may add strings that are regular expressions to the PARTITION QosPolicy. A PARTITION.name is a regular expression if it contains any of the following unescaped special characters: `*`, `?`, `[`, `]`, `!`, or `^`. The PARTITION.name strings can be "concrete" names or regular expression strings; a PARTITION.name element that is a regular expression will only match against concrete strings found in a 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 5.4. 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 5.4: 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 5.5.

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

Table 5.5: 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.

5.9.4 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 5.6.

Table 5.6: Example of Using Location-Based Partitions

Publisher Partitions	Subscriber Partitions	Result
Specify a single partition name using the pattern: "<country>/<state>/<city>"	Specify multiple partition names, one per region of interest	Limits the visibility of the data to *Subscribers* that express interest in the geographical region.
"USA/California/Santa Clara"	(*Subscriber* partition is irrelevant here.)	Send only information for Santa Clara, California.
(Publisher partition is irrelevant here.)	"USA/California/Santa Clara"	Receive only information for Santa Clara, California.
(Publisher partition is irrelevant here.)	"USA/California/Santa Clara" "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 5.7

Table 5.7: Example of Access-Control Group Partitions

Publisher Partitions	Subscriber Partitions	Result
Specify several partition names, one per group that is allowed access:	Specify multiple partition names, one per group to which the *Subscriber* belongs.	Limits the visibility of the data to *Subscribers* that belong to the access-groups specified by the *Publisher*.
"payroll" "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.

5.9.5 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.

5.9.6 Resource limits

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

- `max_partitions`: sets the maximum number of partitions for each PARTITION QoS.
- `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.

5.10 Generating Type Support with rtiddsgen

5.10.1 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.

5.10.2 IDL Type Definition

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

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

For further reference, see the section on Creating User Data Types with IDL in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

5.10.3 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.

5.10.4 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 =
    DDS_DomainParticipant_create_topic(application->participant,
```

(continues on next page)

(continued from previous page)

```
                                        "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.

5.10.5 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.

5.10.6 Unsupported Features of rtiddsgen with Connext Micro

Connext Micro supports a subset of the features and options in *rtiddsgen*. Use `rtiddsgen -micro -help` to see the list of features supported by *rtiddsgen* for *Connext Micro*.

5.11 Threading Model

5.11.1 Introduction

This section describes the threading model, the use of critical sections, and how to configure thread parameters in *RTI Connext Micro*. Please note that the information contained in this document applies to application development using *Connext Micro*. For information regarding *porting* the *Connext Micro* thread API to a new OS, please refer to *Porting Connext Micro*.

This section includes:

- *Architectural Overview*
- *Threading Model*
- *UDP Transport Threads*

5.11.2 Architectural Overview

RTI Connext Micro consists of a core library and a number of components. The core library provides a porting layer, frequently used data-structures and abstractions, and the DDS API. Components provide additional functionality such as UDP communication, DDS discovery plugins, DDS history caches, etc.

```
+-------+                                      \
| 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
```

5.11.3 Threading Model

RTI Connext Micro is architected in a way that makes it possible to create a port of *Connext Micro* that uses no threads, for example on platforms with no operating system. 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*, DomainParticipant-Factory_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 = ....

/* 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.

5.11.4 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.

5.12 Batching

This section is organized as follows:

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

5.12.1 Overview

Batching refers to a mechanism that allows *RTI Connext Micro* to collect multiple user data DDS samples to be sent in a single network packet, to take advantage of the efficiency of sending larger packets and thus increase effective throughput.

Connext Micro supports receiving batches of user data DDS samples, but does not support any mechanism to collect and send batches of user data.

Receiving batches of user samples is transparent to the application, which receives the samples as if the samples had been received one at a time. Note though that the reception sequence number refers to the sample sequence number, not the RTPS sequence number used to send RTPS messages. The RTPS sequence number is the batch sequence number for the entire batch.

A *Connext Micro DataReader* can receive both batched and non-batched samples.

For a more detailed explanation, please refer to the BATCH QosPolicy section in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

5.12.2 Interoperability

RTI Connext Professional supports both sending and receiving batches, whereas *RTI Connext Micro* supports only receiving batches. Thus, this feature primarily exists in *Connext Micro* to interoperate with *RTI Connext* applications that have enabled batching. An *Connext Micro DataReader* can receive both batched and non-batched samples.

5.12.3 Performance

The purpose of batching is to increase throughput when writing small DDS samples at a high rate. In such cases, throughput can be increased several-fold, approaching much more closely the physical limitations of the underlying network transport.

However, collecting DDS samples into a batch implies that they are not sent on the network immediately when the application writes them; this can potentially increase latency. But, if the application sends data faster than the network can support, an increased proportion of the network's available bandwidth will be spent on acknowledgements and DDS sample resends. In this case, reducing that overhead by turning on batching could decrease latency while increasing throughput.

5.12.4 Example Configuration

This section includes several examples that explain how to enable batching in *RTI Connext Professional*. For more detailed and advanced configuration, please refer to the RTI Connext DDS Core Libraries User's Manual.

- This configuration ensures that a batch will be sent with a maximum of 10 samples:

```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:

```
<datawriter_qos>
    <publication_name>
        <name>HelloWorldDataWriter</name>
    </publication_name>
    <batch>
        <enable>true</enable>
        <max_flush_delay>
            <sec>1</sec>
            <nanosec>0</nanosec>
        </max_flush_delay>
    </batch>
</datawriter_qos>
```

- The following configuration ensures that a batch is flushed automatically when max_data_bytes is reached (in this example 8192).

```
<datawriter_qos>
    <publication_name>
        <name>HelloWorldDataWriter</name>
    </publication_name>
    <batch>
        <enable>true</enable>
        <max_data_bytes>8192</max_data_bytes>
    </batch>
</datawriter_qos>
```

Note that max_data_bytes does not include the metadata associated with the batch samples.

Batches must contain whole samples. If a new batch is started and its initial sample causes the serialized size to exceed max_data_bytes, *RTI Connext Professional* will send the sample in a single batch.

5.13 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*

5.13.1 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.

5.13.2 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();

....
```

5.13.3 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.

5.14 Zero Copy Transfer Over Shared Memory

This section is organized as follows:

- *Overview*
- *Getting Started*
- *Synchronization of Zero Copy Samples*
- *Caveats*
- *Further Information*

5.14.1 Overview

Zero Copy transfer over shared memory allows large samples to be transmitted with a minimum number of copies. These 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 shared different memory spaces, A* and B* will be different but they will point to the same place in RAM.

This feature requires the usage of new RTI DDS Extension APIs:

- **FooDataWriter_get_loan()**
- **FooDataWriter_discard_loan()**
- **FooDataReader_is_data_consistent()**

For detailed information, see the C API Reference and C++ API Reference.

5.14.2 Getting Started

To enable Zero Copy transfer over shared memory, 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
   };
   ```

2. Register the Shared Memory Transport (see *Registering the SHMEM Transport*). References will be sent across the shared memory transport.
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
```

Writer Side

Best practice for writing 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,
     * the middleware owns the sample
     */
    dds_rc = FooDataWriter_write(hw_datawriter, sample, &DDS_HANDLE_NIL);
}
```

Reader Side

```
DDS_ReturnCode_t dds_rc;
dds_rc = FooDataReader_take(...)

/* process sample here */

dds_rc = FooDataReader_is_data_consistent(hw_reader,
                                           &is_data_consistent,
                                           sample,sample_info);
```

(continues on next page)

(continued from previous page)

```
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.
         */
     }
}
```

5.14.3 Synchronization of Zero Copy Samples

There is **NO** synchronization of a zero copy sample between a sender (*DataWriter*) and receiver (*DataReader*) application. It is possible for a sample's content to be invalidated before the receiver application actually has had a chance to read it.

To illustrate this scenario, consider creating the case of creating a *Best-effort DataWriter* with max_samples of **X=1**. When the *DataWriter* is created the middleware will pre-allocate a pool of **X+1** (2) samples residing in a shared memory region. This pool will be used to loan samples when calling **FooDataWriter_get_loan(...)** ,

```
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);
/*
 * Because the datawriter is using best effort, the middleware immediately
 * makes this sample available to be returned by another FooDataWriter_get_loan(...
 */

ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 2 */
sample->value = 20000;
ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);
/*
 * Because the datawriter is using best effort, the middleware immediately
 * makes this sample available to be returned by another FooDataWriter_get_loan(...
 */

/*
 * At this point, it is possible the sample has been received by the receiving␣
↪application
 * but has not been presented yet to the user.
 */
```

(continues on next page)

(continued from previous page)

```
ddsrc = FooDataWriter_get_loan(dw, &sample); /* returns pointer to sample 1 */
/* sample->value will contain the integer 10000 because we are re-using samples
 * from a list that contains only 2 buffers.
 *
 * Also, at this point in time a referemce to sample 1 and 2 may have already been␣
↪received
 * by the middleware on the DataReader side and are lying inside a DataReader's internal␣
↪cache.
 * However, the sample may not have been received by the
 * application. If at this point the sample's value (sample->value) was changed to 999,
 * the sample returned from the Subscribers
 * read(...) or take(...) would contain unexpected values (999 instead of 10000). This␣
↪is because
 * both the Publisher and the Subscriber process have mapped into their virtual
 * address space the same shared memory region where the sample lies.
 *
 * Use **FooDataReader_is_data_consistent** to verify the consistency, to prevent this
 * scenario.
 *
 * Note, a sample is actually invalidated right after the completion
 * of FooDataWriter_get_loan(dw, &sample). If the address of the newly created sample␣
↪has been
 * previously written and its contents has not been read by the receiver application,
 * then the previously written sample has been invalidated.
 */

ddsrc = FooDataWriter_write(datawriter, sample, &DDS_HANDLE_NIL);
```

5.14.4 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 FooDataWriter. Otherwise, subsequent **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.

5.14.5 Further Information

For more information, see the section on Zero Copy Transfer Over Shared Memory in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

5.15 FlatData Language Binding

This section is organized as follows:

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

5.15.1 Overview

RTI Connext Micro supports the FlatData^{TM} 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.

5.15.2 Getting Started

The best way to start is to generate an example by creating an example IDL file HelloWorld.idl containing the following IDL type:

```
@final
@language_binding(FLAT_DATA)
struct HelloWorld
{
   long a;
}
```

Next, run:

```
rtiddsgen -example -micro -language C++ HelloWorld.idl
```

5.15.3 Further Information

For more details about this feature, please see the FlatData Language Binding section in the RTI Connext DDS Core Libraries User's Manual (available here if you have Internet access).

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

5.16 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). See *Examples*.

5.16.1 Defining an Application in XML

Each *Entity* configured in the XML file is given a name. This name is used to retrieve the entities at runtime using the *Connext Micro* API.

In the XML file, you need to distinguish between two names:

- Configuration name: The name of a specific *Entity*'s configuration. It is given by the name attribute of the corresponding element.
- Entity name in the *Entity*'s QoS: The Entity name in the *Entity*'s QoS.

At runtime, the *Entity* will be created using the Entity name in the *Entity*'s QoS; the configuration name will be used if this is an empty string.

The attribute multiplicity indicates that a set of *Entities* should be created from the same configuration. Since each *Entity* must have a unique name, the system will automatically append a number to the Entity name in the *Entity*'s QoS (or, if it is an empty string, the configuration name) to obtain the Entity name. For example, if we specified a multiplicity of "N", then for each index "i" between 0 and N-1, the system will assign Entity names according to the table below:

Entity Name	Index: i
"configuration_name"	0
"configuration_name#i"	[1,N-1]

That is, the *Entity* name followed by the token "#" and an index.

See *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.

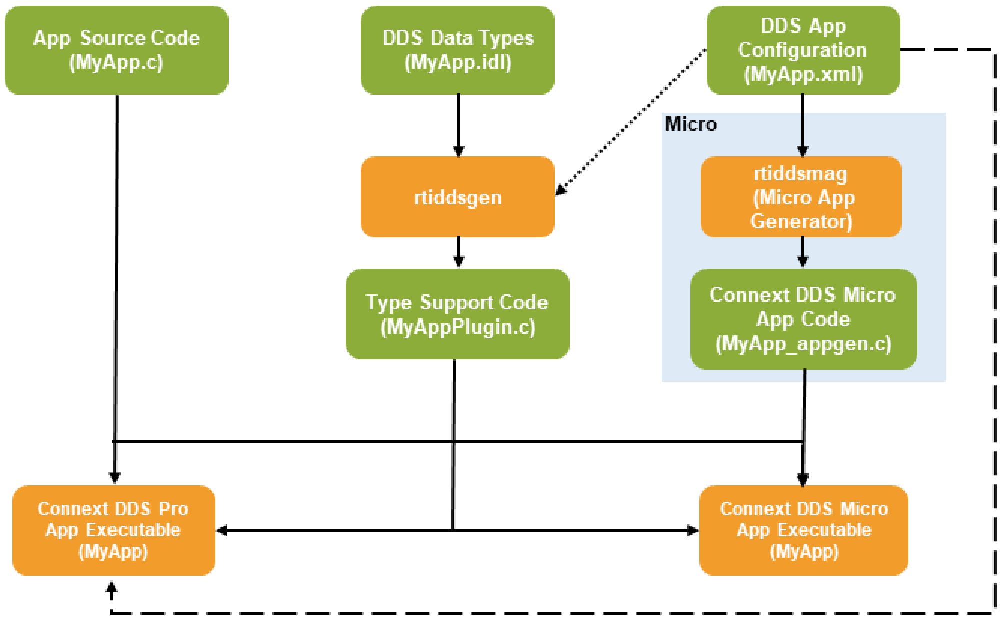
Changes in the XML configuration file require regenerating the C/C++ source code and recompiling the application.

5.16.2 Generating the Application from XML

Connext Micro comes with a tool, the Micro Application Generator (MAG). This tool is used to generate supporting files to create XML-defined applications at runtime.

Micro Application Generator (MAG) Tool

Micro Application Generator (MAG) is a required tool to configure *Connext Micro* applications by generating code from an XML configuration file; it creates DDS entities and registers all the components needed for a *Connext Micro*-based application. MAG can process your own XML configuration file, or it can process an *XML-Based Application Creation* file that you created for *RTI Connext Professional*.


App Source Code (MyApp.c)
DDS Data Types (MyApp.idl)
DDS App Configuration (MyApp.xml)
Micro
rtiddsgen
rtiddsmag (Micro App Generator)
Type Support Code (MyAppPlugin.c)
Connext DDS Micro App Code (MyApp_appgen.c)
Connext DDS Pro App Executable (MyApp)
Connext DDS Micro App Executable (MyApp)

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 5.8: 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 5.9: 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_config()** 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.

5.16.3 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.

5.16.4 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 provided when you install Connext Micro. You will find them in the directory **<path to Micro examples>/C/HelloWorld_appgen**. The following examples are provided:

- **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. You can find this file in **<path to Micro examples>/C/HelloWorld_appgen**.

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

(continues on next page)

(continued from previous page)

```
    </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 use the C language in this example.

To generate code with rtiddsgen:

- On a Windows system:

  From your command shell, change directory to **<path to Micro examples>CHelloWorld_appgen** and type:

  ```
  <RTIMEHOME>\rtiddsgen\scripts\rtiddsgen.bat -language C HelloWorld.xml
  ```

  Note: The Visual Studio solution in the example folder automatically calls *rtiddsgen*.

- On a Linux or macOS system:

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

  ```
  <RTIMEHOME>/rtiddsgen/scripts/rtiddsgen -language C HelloWorld.xml
  ```

After running *rtiddsgen*, you will see the following files in the **HelloWorld_appgen** directory:

- **HelloWorld.h**
- **HelloWorld.c**
- **HelloWorldPlugin.h**
- **HelloWorldPlugin.c**
- **HelloWorldSupport.h**
- **HelloWorldSupport.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 use C in this 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>CHelloWorld_appgen** and type:

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

Note: The Visual Studio solution in the example folder automatically calls rtiddsmag.

- On a Linux or macOS system:

  From your command shell, change directory to **<path to Micro examples>/C/HelloWorld_appgen** 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*.

```xml
<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

```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, /* flow_controller_count */ \
    NULL /* 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*.

```
<?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 and SHMEM) 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.
- DPDE: **dpde** + participant_number.
- DPSE: **dpse** + participant_number.

These names can be changed by using the **...Names** 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 */ \
        0L /* nanosec */ \
    }, \
```

(continues on next page)

(continued from previous page)

```
    {   /*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 throw 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 *DomainPar-*

ticipant, *DataWriter* or *DataReader*, you could use the transport names for the built-in transport plugins: **shmem** and **udpv4**. *rtiddsmag* will automatically modify this alias to match the new one with the *DomainParticipant* number at the end of the name.

Flow Controllers

rtiddsmag creates code which will be used by *Connext Micro* to create a flow controller. The flow controller is configured through properties in the XML file. Let's see an example of how to configure a 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>
            <element>
                <name>
  dds.flow_controller.token_bucket.custom_flowcontroller.token_bucket.bytes_per_token
                </name>
```

(continues on next page)

(continued from previous page)

```
                <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_FlowControllerModel
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 */ \
}

#define
RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPDEPubDP_HelloWorldDPDEPub_
↪HelloWorldDPDEDW \
```

(continues on next page)

(continued from previous page)

```
{ \
    ...
    {   /* publish_mode */ \
        DDS_ASYNCHRONOUS_PUBLISH_MODE_QOS, /* max_remote_readers */ \
        "custom_flowcontroller", /* flow_controller_name */ \
        12L /* priority */ \
    }, \
    ...
}

extern const struct APPGEN_FlowControllerModel
HelloWorldAppLibrary_HelloWorldDPDEPubDP_flow_controllers[1];

#define RTI_APP_GEN__DP_HelloWorldAppLibrary_HelloWorldDPDEPubDP \
{ \
    ...
    1UL, /* flow_controller_count */ \
    HelloWorldAppLibrary_HelloWorldDPDEPubDP_flow_controllers /* flow_controllers */ \
}
```

Static Discovery

rtiddsmag iterates through each *DomainParticipant* definition in the XML configuration file, creating the remote entities that are needed to communicate with applications that use static discovery, and updating the **object__id** of each *DataWriter* or *DataReader* involved if they don't have a valid value or they are using the default value.

Let's see an example of two applications that use static discovery and how *rtiddsmag* generates the necessary code that will be asserted by *Connext Micro* to communicate with both applications:

```
<domain_participant name="HelloWorldDPSEPubDP"
    domain_ref="HelloWorldLibrary::HelloWorldDomain">
    <publisher name="HelloWorldDPSEPub">
        <data_writer topic_ref="HelloWorldTopic" name="HelloWorldDPSEDW">
            <datawriter_qos base_name="QosLibrary::DPSEProfile"/>
        </data_writer>
    </publisher>
    <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
</domain_participant>

<domain_participant name="HelloWorldDPSESubDP"
    domain_ref="HelloWorldLibrary::HelloWorldDomain">
    <subscriber name="HelloWorldDPSESub">
        <data_reader topic_ref="HelloWorldTopic" name="HelloWorldDPSEDR">
          <datareader_qos base_name="QosLibrary::DPSEProfile"/>
        </data_reader>
    </subscriber>
    <domain_participant_qos base_name="QosLibrary::DPSEProfile"/>
</domain_participant>
```

For these two *DomainParticipants*, *rtiddsmag* will update the **rtps_object_id** for the *DataWriter* and *DataReader*, since they didn't have any values set in the XML file. You can see this in the following snippet from **HelloWorldAppgen.h**:

```
#define
RTI_APP_GEN___DW_QOS_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldDPSEPub_
↪HelloWorldDPSEDW \
{ \
    ...
    {   /* protocol */ \
        1UL, /* rtps_object_id */ \
        { /* rtps_reliable_writer */ \
            {   /* heartbeat_period */ \
                0L, /* sec */ \
                250000000UL /* nanosec */ \
            },  \
            1L, /* heartbeats_per_max_samples */ \
            DDS_LENGTH_UNLIMITED, /* max_send_window */ \
            DDS_LENGTH_UNLIMITED, /* max_heartbeat_retries */ \
            {   /* first_write_sequence_number */ \
                0, /* high */ \
                1  /* low */ \
            } \
        }, \
        DDS_BOOLEAN_TRUE /* serialize_on_write */ \
    }, \
    ...
}

#define
RTI_APP_GEN___DR_QOS_HelloWorldAppLibrary_HelloWorldDPSESubDP_HelloWorldDPSESub_
↪HelloWorldDPSEDR \
{ \
    ...
    {   /* protocol */ \
        2UL /* rtps_object_id */ \
    }, \
    ...
}
```

rtiddsmag will also generate the remote *DomainParticipants*, *DataWriters*, and *DataReaders* that need to be asserted in order for endpoints to match:

HelloWorldAppgen.c

```
const struct APPGEN_RemoteSubscriptionModel
HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_subscribers[1] =
{
    RTI_APP_GEN__RSD_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_HelloWorldDPSESub_HelloWorldDPSEDR
};

const struct APPGEN_RemoteParticipantModel
```

(continues on next page)

(continued from previous page)

```
HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_participants[1] =
{
    {
        "HelloWorldDPSESubDP", /* name */
        0UL,  /* remote_publisher_count */
        NULL, /* remote_publishers */
        1UL,  /* remote_subscriber_count */
        HelloWorldAppLibrary_HelloWorldDPSEPubDP_remote_subscribers /* remote_
↪subscribers */
    }
};

const struct APPGEN_RemotePublicationModel
HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_publishers[1] =
{
    RTI_APP_GEN__RPD_HelloWorldAppLibrary_HelloWorldDPSESubDP_HelloWorldAppLibrary_
↪HelloWorldDPSEPubDP_HelloWorldDPSEPub_HelloWorldDPSEDW
};

const struct APPGEN_RemoteParticipantModel
HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_participants[1] =
{
    {
        "HelloWorldDPSEPubDP", /* name */
        1UL, /* remote_publisher_count */
        HelloWorldAppLibrary_HelloWorldDPSESubDP_remote_publishers, /* remote_publishers␣
↪*/
        0UL, /* remote_subscriber_count */
        NULL /* remote_subscribers */
    }
};
```

HelloWorldAppgen.h

```
#define RTI_APP_GEN__RSD_HelloWorldAppLibrary_HelloWorldDPSEPubDP_HelloWorldAppLibrary_
↪HelloWorldDPSESubDP_HelloWorldDPSESub_HelloWorldDPSEDR \
{ \
    { /* subscription_data */ \
        { \
            { 0, 0, 0, 2 } /* key */ \
        }, \
        { \
            { 0, 0, 0, 0 } /* participant_key */ \
        }, \
        "HelloWorldTopic", /* topic_name */ \
        "HelloWorldType",  /* type_name */ \
        DDS_DEADLINE_QOS_POLICY_DEFAULT, \
        DDS_OWNERSHIP_QOS_POLICY_DEFAULT, \
        DDS_LATENCY_BUDGET_QOS_POLICY_DEFAULT, \
        {   /* reliability */ \
            DDS_RELIABLE_RELIABILITY_QOS, /* kind */ \
```

(continues on next page)

(continued from previous page)

```
            {   /* max_blocking_time */ \
                0L, /* sec */ \
                0L  /* nanosec */ \
            } \
        }, \
        DDS_LIVELINESS_QOS_POLICY_DEFAULT, \
        DDS_DURABILITY_QOS_POLICY_DEFAULT, \
        DDS_DESTINATION_ORDER_QOS_POLICY_DEFAULT, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_SEQUENCE_INITIALIZER, \
        DDS_DATA_REPRESENTATION_QOS_POLICY_DEFAULT \
        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,  /* flow_controller_count */ \
    NULL, /* 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␣
↪*/ \
```

(continues on next page)

(continued from previous page)

```
    0UL, /* flow_controller_count */ \
    NULL /* 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 */
↪ \
```

(continues on next page)

(continued from previous page)

```
    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,  /* flow_controller_count */ \
    NULL, /* 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];
```

(continues on next page)

(continued from previous page)

```
#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, /* flow_controller_count */ \
    NULL /* flow_controllers */ \
}
```

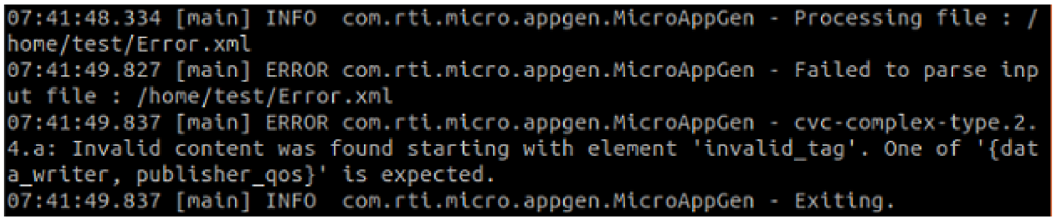
5.16.5 Errors Caused by Invalid Configurations and QoS

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:

  ```
  <dds>
      ...
      <!-- Participant Library -->
      <domain_participant_library name="FeatureTestLibrary">
          <domain_participant name="01_EmptyDomainParticipant"
          domain_ref="HelloWorldLibrary::HelloWorldDomain">
              <invalid_tag></invalid_tag>
          </domain_participant>
      </domain_participant_library>
      ...
  </dds>
  ```

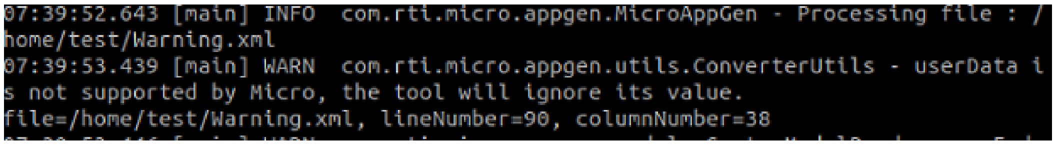


```
07:41:48.334 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Error.xml
07:41:49.827 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to parse inp
ut file : /home/test/Error.xml
07:41:49.837 [main] ERROR com.rti.micro.appgen.MicroAppGen - cvc-complex-type.2.
4.a: Invalid content was found starting with element 'invalid_tag'. One of '{dat
a_writer, publisher_qos}' is expected.
07:41:49.837 [main] INFO  com.rti.micro.appgen.MicroAppGen - Exiting.
```

- **Unsupported elements**

  MAG will throw a warning for any elements that are not supported by *Connext Micro*. Unsupported elements will be ignored, such as the user_data in the following:

```
<dds>
    ...
    <!-- Participant Library -->
    <domain_participant_library name="FeatureTestLibrary">
        <domain_participant name="01_EmptyDomainParticipant"
        domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <domain_participant_qos>
                <!-- user_data is not supported by Micro -->
                <user_data/>
            </domain_participant_qos>
        </domain_participant>
    </domain_participant_library>
</dds>
```



```
07:39:52.643 [main] INFO  com.rti.micro.appgen.MicroAppGen - Processing file : /
home/test/Warning.xml
07:39:53.439 [main] WARN  com.rti.micro.appgen.utils.ConverterUtils - userData i
s not supported by Micro, the tool will ignore its value.
file=/home/test/Warning.xml, lineNumber=90, columnNumber=38
```

- **Unsupported values**

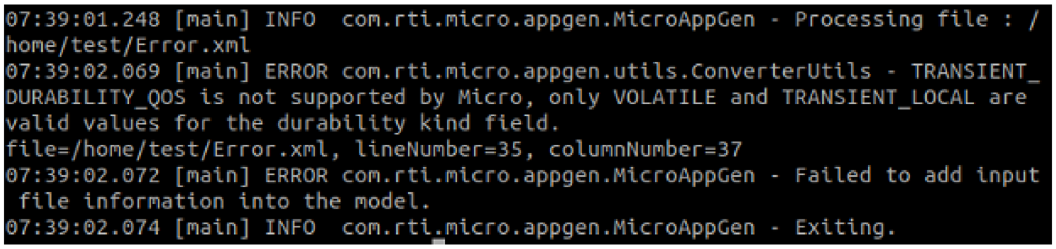
  MAG will throw an error if it finds a value that is not supported by *Connext Micro*.

```
<dds>
    ...
    <!-- Participant Library -->
    <domain_participant_library name="FeatureTestLibrary">
        <domain_participant name="01_EmptyDomainParticipant"
        domain_ref="HelloWorldLibrary::HelloWorldDomain">
            <publisher name ="test">
                <data_writer topic_ref="HelloWorldTopic1" name="testW">
                    <datawriter_qos>
                        <durability>
                            <!-- transient is not supported by Micro -->
                            <kind>TRANSIENT_DURABILITY_QOS</kind>
                        </durability>
                    </datawriter_qos>
                </data_writer>
            </publisher>
        </domain_participant>
```

(continues on next page)

(continued from previous page)

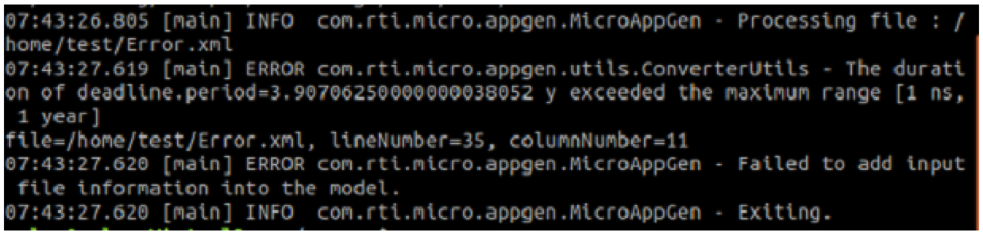
```
    </domain_participant_library>
</dds>
```



```
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.
```

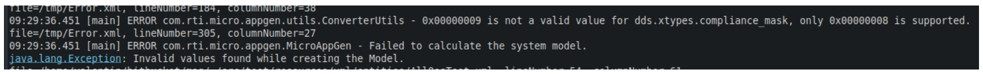
MAG will throw an error if the QoS values are not consistent with values supported in *Connext Micro*. For example, the following XML contains a deadline period that is too large.

```
<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=3.90706250000000038052 y exceeded the maximum range [1 ns,
 1 year]
file=/home/test/Error.xml, lineNumber=35, columnNumber=11
07:43:27.620 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to add input
 file information into the model.
07:43:27.620 [main] INFO  com.rti.micro.appgen.MicroAppGen - Exiting.
```

MAG will throw an error if the **dds.xtypes.compliance_mask** property uses a different value than 0x00000008.



```
file=/tmp/Error.xml, lineNumber=184, columnNumber=38
09:29:36.451 [main] ERROR com.rti.micro.appgen.utils.ConverterUtils - 0x00000009 is not a valid value for dds.xtypes.compliance_mask, only 0x00000008 is supported.
file=/tmp/Error.xml, lineNumber=305, columnNumber=27
09:29:36.451 [main] ERROR com.rti.micro.appgen.MicroAppGen - Failed to calculate the system model.
java.lang.Exception: Invalid values found while creating the Model.
```

- **Unsupported QoS**

  Not all the QoS policies supported by *Connext Micro* can be configured in XML.

  - QoS settings related to UDP transformation cannot be configured in XML. See the *UDP Transport* section for more information on UDP transformation.
  - MAG does not support any PROPERTY QoS policy properties except the **dds.xtypes.compliance_mask** property.

5.17 Building Against FACE Conformance Libraries

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

5.17.1 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.

5.17.2 FACE Golden Libraries

The FACE conformance tools use a set of golden libraries. There are different golden libraries for different FACE services, languages and profiles. *Connext Micro only* conforms to the safetyExt and safety profile of 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.

5.17.3 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>**.

5.18 Working With Sequences

5.18.1 Introduction

RTI Connext Micro uses IDL as the language to define data-types. One of the constructs in IDL is the *sequence*: a variable-length vector where each element is of the same type. This section describes how to work with sequences; in particular, the string sequence since it has special properties.

Note: This section references several sequence APIs supported by *Connext Micro*. However, *Connext Cert* only supports a subsection of these APIs. Please refer to Sequence Support in the C

API Reference for a full list; the APIs supported by *Connext Cert* will have a **«cert»** annotation in their description.

5.18.2 Working with Sequences

Overview

Logically a sequence can be viewed as a variable-length vector with N elements, as illustrated below. Note that sequences indices are 0 based.

```
     +---+
  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");
}

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);
```

5.19 Debugging

5.19.1 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.

5.19.2 Configuring Logging

By default, *Connext Micro* sets the log verbosity to *Error*. It can be changed at any time by calling **OSAPI_Log_set_verbosity()** using the desired verbosity as a parameter.

Note 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.

5.19.3 Log Message Kinds

Each log entry is classified as one of the following kinds:

- *Error*. An unexpected event with negative functional impact.
- *Warning*. An event that may not have negative functional impact but could indicate an unexpected situation.
- *Information*. An event logged for informative purposes.

By default, the log verbosity is set to *Error*, so only error logs will be visible. To change the log verbosity, simply call the function **OSAPI_Log_set_verbosity()** with the desired verbosity level.

5.19.4 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)".

Chapter 6

Porting Connext Micro

RTI Connext Micro has been engineered for reasonable portability to platforms and environments which RTI does not have access to. This porting guide describes the features required by *Connext Micro* to run. The target audience is developers familiar with general OS concepts, the standard C library, and embedded systems.

Connext Micro uses an abstraction layer to support running on a number of platforms. The abstraction layer, OSAPI, is an abstraction of functionality typically found in one or more of the following libraries and services:

- Operating System calls
- Device drivers
- Standard C library

The OSAPI module is designed to be relatively easy to move to a new platform. All functionality, with the exception of the UDP transport which must be ported, is contained within this single module. It should be noted that although some functions may not seem relevant on a particular platform, they must still be implemented as they are used by other modules. For example, the port running on Stellaris with no OS support still needs to implement a threading model.

Please note that the OSAPI module is not designed to be a general purpose abstraction layer; its sole purpose is to support the execution of *Connext Micro*.

6.1 Updating from Connext Micro 2.4.8 and earlier

In *RTI Connext Micro* 2.4.9, a few changes were made to simplify incorporating new ports. To upgrade an existing port to work with 2.4.9, follow these rules:

- Any changes to osapi_config.h should be placed in its own file (see *Directory Structure*).
- Define the OSAPI_OS_DEF_H preprocessor directive to include the file (refer to *OS and CC Definition Files*).
- For compiler-specific definitions, please refer to *OS and CC Definition Files*.
- Please refer to *Heap Porting Guide* for changes to the Heap routines that need to be ported.

6.2 Directory Structure

The source shipped with *Connext Micro* is identical to the source developed and tested by RTI (with the exception of the the line-endings difference between the Unix and Windows source-bundles).

The source-bundle directory structure is as follows:

```
RTIMEHOME--+-- CmakeLists.txt
           |
           +-- build -- cmake --+-- Debug --+-- <ARCH> -- <project-files>
           |                    |
           |                    |
           |                    +-- Release --+-- <ARCH> -- <project-files>
           +-- doc --
           |
           +-- example
           |
           +-- include
           |
           +-- lib +-- <ARCH> -- <libraries>
           |
           +-- resource --+-- cmake
           |              |
           |              +-- scripts
           |
           +-- rtiddsgen
           |
           +-- rtiddsmag
           |
           +-- src
```

The include directory contains the external interfaces, those that are available to other modules. The src directory contains the implementation files. Please refer to *Building the Source for Common Platforms* for how to build the source code.

The remainder of this document focuses on the files that are needed to add a new port. The following directory structure is expected:

```
---+-- include --+-- osapi --+-- osapi_os_\<port\>.h
   |                         |
   |                         +-- osapi_cc_<compiler>.h
   |
   +-- src --+-- osapi --+-- common -- <common files>
                         |
                         +-- <port> --+-- <port>Heap.c
                                      |
                                      +-- <port>Mutex.c
                                      |
                                      +-- <port>Process.c
                                      |
                                      +-- <port>Semaphore.c
                                      |
```

(continues on next page)

(continued from previous page)

```
                                    +-- <port>String.c
                                    |
                                    +-- <port>System.c
                                    |
                                    +-- <port>Thread.c
                                    |
                                    +-- <port>shmSegment.c
                                    |
                                    +-- <port>shmMutex.c
```

The *osapi_os_<port>.h* file contains OS specific definitions for various data-types. The <port> name should be short and in lower case, for example *myos*.

The *osapi_cc_<compiler>.h* file contains compiler specific definitions. The <compiler> name should be short and in lower case, for example *mycc*. The osapi_cc_stdc.h file properly detects GCC and MSVC and it is not necessary to provide a new file if one of these compilers is used.

The <port>Heap.c, <port>Mutex.c, <port>Process.c, <port>Semaphore.c, <port>String.c and <port>System.c files shall contain the implementation of the required APIs.

NOTE: It is *not* recommended to modify source files shipped with *Connext Micro*. Instead if it is desired to start with code supplied by RTI it is recommended to *copy* the corresponding sub-directory, for example posix, and rename it. This way it is easier to upgrade *Connext Micro* while keeping existing ports.

6.3 OS and CC Definition Files

The *include/osapi/osapi_os_<port>.h* file contains OS and platform specific definitions used by OSAPI and other modules. To include the platform specific file, define **OSAPI_OS_DEF_H** as a preprocessor directive.

```
-DOSAPI_OS_DEF_H=\"osapi_os_<port>.h\"
```

It should be noted that *Connext Micro* does not use auto-detection programs to detect the host and target build environment and only relies on predefined macros to determine the target environment. If *Connext Micro* cannot determine the target environment, it is necessary to manually configure the correct OS definition file by defining **OSAPI_OS_DEF_H** (see above).

The *include/osapi/osapi_cc_<compiler>.h* file contains compiler specific definitions used by OSAPI and other modules. To include the platform specific file, define **OSAPI_CC_DEF_H** as a preprocessor directive.

```
-DOSAPI_CC_DEF_H=\"osapi_cc_<compiler>.h\"
```

Endianness of some platforms is determined automatically via the platform specific file, but for others either **RTI_ENDIAN_LITTLE** or **RTI_ENDIAN_BIG** must be defined manually for little-endian or big-endian, respectively.

6.4 Heap Porting Guide

Connext Micro uses the heap to allocate memory for internal data-structures. With a few exceptions, *Connext Micro* does *not* return memory to the heap. Instead, *Connext Micro* uses internal pools to quickly allocate and free memory for specific types. Only the initial memory is allocated directly from the heap. The following functions must be ported:

- OSAPI_Heap_allocate_buffer
- OSAPI_Heap_free_buffer

However, if the OS and C library supports the standard malloc and free APIs define the following in the *osapi_os_<port>.h* file:

```
#define OSAPI_ENABLE_STDC_ALLOC   (1)
#define OSAPI_ENABLE_STDC_REALLOC (1)
#define OSAPI_ENABLE_STDC_FREE    (1)
```

Please refer to the OSAPI_Heap API for definition of the behavior. The available source code contains implementation in the file *osapi/<port>/<port>Heap.c*.

6.5 Mutex Porting Guide

Connext Micro relies on mutex support to protect internal data-structures from corruption when accessed from multiple threads.

The following functions must be ported:

- OSAPI_Mutex_new
- OSAPI_Mutex_delete
- OSAPI_Mutex_take_os
- OSAPI_Mutex_give_os

Please refer to the OSAPI_Mutex API for definition of the behavior. The available source code contains implementation in the file *osapi/<port>/<port>Mutex.c*

6.6 Semaphore Porting Guide

Connext Micro relies on semaphore support for thread control. If *Connext Micro* is running on a non pre-emptive operating system with no support for IPC and thread synchronization, it is possible to implement these functions as no-ops. Please refer to *Thread Porting Guide* for details regarding threading.

The following functions must be ported:

- OSAPI_Semaphore_new
- OSAPI_Semaphore_delete

- OSAPI_Semaphore_take
- OSAPI_Semaphore_give

Please refer to the OSAPI_Semaphore API for definition of the behavior. The available source code contains implementation in the file *osapi/<port>/<port>Semaphore.c*.

6.7 Process Porting Guide

Connext Micro only uses the process API to retrieve a unique ID for the applications.

The following functions must be ported:

- OSAPI_Process_getpid

Please refer to the OSAPI_Process_getpid API for definition of the behavior. The available source code contains implementation in the file *osapi/<port>/<port>Process.c*.

6.8 System Porting Guide

The system API consists of functions which are more related to the hardware on which *Connext Micro* is running than on the operating system. As of *Connext Micro* 2.3.1, the system API is implemented as an interface as opposed to the previous pure function implementation. This change makes it easier to adapt *Connext Micro* to different hardware platforms without having to write a new port.

The system interface is defined in OSAPI_SystemI, and a port must implement all the methods in this structure. In addition, the function OSAPI_System_get_native_interface must be implemented. This function must return the system interface for the port (called the native system interface).

The semantics for the methods in the interface are exactly as defined by the corresponding system function. For example, the method OSAPI_SystemI::get_time must behave exactly as that described by OSAPI_System_get_time.

The following system interface methods must be implemented in the OSAPI_SystemI structure:

- OSAPI_SystemI::get_timer_resolution
- OSAPI_SystemI::get_time
- OSAPI_SystemI::start_timer
- OSAPI_SystemI::stop_timer
- OSAPI_SystemI::generate_uuid
- OSAPI_SystemI::get_hostname
- OSAPI_SystemI::initialize
- OSAPI_SystemI::finalize

Please refer to the OSAPI_System API for definition of the behavior. The available source code contains implementation in the file: *osapi/<port>/<port>System.c*.

6.8.1 Migrating a 2.2.x port to 2.3.x

In *Connext Micro* 2.3.x, changes where made to how the system API is implemented. Because of these changes, existing ports must be updated, and this section describes how to make a *Connext Micro* 2.2.x port compatible with *Connext Micro* 2.3.x.

If you have ported *Connext Micro* 2.2.x the following steps will make it compatible with version 2.3.x:

- Rename the following functions and make them private to your source code. For example, rename OSAPI_System_get_time to OSAPI_MyPortSystem_get_time etc.
  - OSAPI_System_get_time
  - OSAPI_System_get_timer_resolution
  - OSAPI_System_start_timer
  - OSAPI_System_stop_timer
  - OSAPI_System_generate_uuid
- Implement the following new methods.
  - OSAPI_SystemI::get_hostname
  - OSAPI_SystemI::initialize
  - OSAPI_SystemI::finalize
- Create a system structure for your port using the following template:

```
 struct OSAPI_MyPortSystem
{
    struct OSAPI_System _parent;

    Your system variable
};

static struct OSAPI_MyPortSystem OSAPI_System_g;

/* OSAPI_System_gv_system is a global system variable used by the
 * generic system API. Thus, the name must be exactly as
 * shown here.
 */
struct OSAPI_System * OSAPI_System_gv_system = &OSAPI_System_g._parent;
```

- Implement OSAPI_System_get_native_interface method and fill the OSAPI_SystemI structure with all the system methods.

6.9 Thread Porting Guide

The thread API is used by *Connext Micro* to create threads. Currently only the UDP transport uses threads and it is a goal to keep the generic *Connext Micro* core library free of threads. Thus, if *Connext Micro* is ported to an environment with no thread support, the thread API can be stubbed out. However, note that the UDP transport must be ported accordingly in this case; that is, all thread code must be removed and replaced with code appropriate for the environment.

The following functions must be ported:

- OSAPI_Thread_create
- OSAPI_Thread_sleep

Please refer to the OSAPI_Thread API for definition of the behavior. The available source code contains implementation in the file *srcC/osapi/<platform>/Thread.c*.

Chapter 7

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*.

7.1 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.

7.2 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.

7.3 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*.

7.4 Standard APIs

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

7.5 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.

7.6 Interoperability

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

7.6.1 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.

7.6.2 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*.

7.7 Connext Tools

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

7.7.1 Admin Console

Admin Console can discover and display *Connext Micro* applications that use full dynamic discovery (DPDE). When using static discovery (DPSE), it is required to use the Limited Bandwidth Endpoint Discovery (LBED) that is available as a separate product for *Connext Professional*. With the library a configuration file with the discovery configuration must be provided (just as in the case for products such as Routing Service, etc.). This is provided through the QoS XML file.

Data can be visualized from *Connext Micro DataWriters*. Keep in mind that *Connext Micro* does not currently distribute type information and the type information has to be provided through an XML file using the "Create Subscription" dialog. Unlike some other products, this information cannot be provided through the QoS XML file. To provide the data types to Admin Console, first run the code generator with the `-convertToXml` option:

```
rtiddsgen -convertToXml <file>
```

Then click on the "Load Data Types from XML file" hyperlink in the "Create Subscription" dialog and add the generated IDL file.

Other Features Supported:

- Match analysis is supported.
- Discovery-based QoS are shown.

The following resource-limits in *Connext Micro* must be incremented as follows when using Admin Console:

- Add 24 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_reader_allocation
- Add 24 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_writer_allocation
- Add 1 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_allocation
- Add 1 to DDS_DomainParticipantResourceLimitsQosPolicy::remote_participant_allocation if data-visualization is used

Connext Micro does not currently support any administration capabilities or services, and does not match with the Admin Console *DataReaders* and *DataWriters*. However, if matching *DataReaders* and *DataWriters* are created, e.g., by the application, the following resource must be updated:

- Add 48 to DDS_DomainParticipantResourceLimitsQosPolicy::matching_writer_reader_pair_allocation

7.7.2 Distributed Logger

This product is not supported by *Connext Micro*.

7.7.3 LabVIEW

The LabVIEW toolkit uses *Connext Professional*, and it must be configured as any other *Connext Professional* application. A possible option is to use the builtin *Connext Professional* profile: BuiltinQosLib::Generic.ConnextMicroCompatibility.

7.7.4 Monitor

This product is not supported by *Connext Micro*.

7.7.5 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.

7.7.6 Wireshark

Wireshark fully supports *Connext Micro*.

7.7.7 Persistence Service

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

7.7.8 Application Generation Using XML

An application defined in XML can be shared between *Connext Micro* and *Connext Professional*, with the limitations documented in *Application Generation Using XML*.

Chapter 8

API Reference

RTI Connext Micro features API support for C and C++. Select the appropriate language below in order to access the corresponding API Reference HTML documentation.

- C API Reference
- C++ API Reference

Chapter 9

Release Notes

9.1 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.

OS	CPU	Compiler	RTI Architecture Abbreviation
Windows 10	x64	VS 2017	x64Win64VS2017
OS X 11	x64	clang 12.0	x64Darwin20clang12.0
Ubuntu 22.04 LTS	x64	gcc 12.3.0	x64Linux4gcc12.3.0
Ubuntu 18.04 LTS	x64	gcc 7.3.0	x64Linux4gcc7.3.0
Ubuntu 18.04 LTS	ARMv8 (64-bit)	gcc 7.3.0	armv8Linux4gcc7.3.0
PPC Linux (Kernel version 3)	ppce500	gcc 4.7.2	ppce500mcLinux3gcc4.7.2
QNX 7.1	armv8	qcc_gpp8.3.0	armv8QNX7.1qcc_gpp8.3.0

9.2 What's New in 4.0.1

The following features are new since *Connext Micro* 4.0.0.

9.2.1 Enable or disable padding bits with PROPERTY QoS policy in DomainParticipant and Data Writer

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.

9.2.2 Generate examples with new template options for Code Generator

In this release, some examples that were previously included in a *Connext Micro* installation have been removed. Instead, examples can be generated from templates included with the *RTI Code Generator*.

This release introduces two new *Code Generator* command-line options, `-showTemplates` and `-exampleTemplates`.

The `-showTemplates` option prints and generates an XML file containing a list of available example templates in your *Connext Micro* installation, organized per language.

The `-exampleTemplate` option generates an example you specify, instead of the default one.

When you use the `-exampleTemplate` option, you can specify one of the example templates in `$RTIMEHOME/rtiddsgen/resource/templates/example/<language>/<templateName>/`. You may also create your own templates and place them in this directory.

You must also use one of the following command-line options:

- `-create examplefiles`
- `-update examplefiles`
- `-example`

Code Generator will then generate the example you specified. For example:

```
rtiddsgen -language C++ -example -exampleTemplate <exampleTemplateName> foo.idl
```

For more information, please refer to *Example Generation*.

9.3 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.

9.3.1 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]

9.3.2 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]

9.3.3 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]

9.3.4 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]

9.3.5 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]

9.3.6 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]

9.3.7 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]

9.4 Previous Releases

9.4.1 What's New in 4.0.0

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>
```

9.4.2 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]

9.5 Known Issues

9.5.1 C++ constructor does not allocate memory for Topic and Type names

The C++ constructor for DDS_PublicationBuiltinTopicData and DDS_SubscriptionBuiltinTopicData does not allocate memory for the **topic_name** and **type_name** attributes. Memory must be allocated using either DDS_String_dup() or another memory allocator, or by using the **DDS_PublicationBuiltinTopicData::initialize_from_qos(const DDS_DomainParticipantQos &dp_qos)** or **DDS_SubscriptionBuiltinTopicData::initialize_from_qos(const DDS_DomainParticipantQos &dp_qos)** methods.

Warning: Note that the **initialize_from_qos()** method is not documented in the C++ API Reference; see *RTI Issue MICRO-7112*. Please refer to the Discovery module in the C API Reference instead.

[RTI Issue ID MICRO-7110]

9.5.2 initialize_from is not documented in C++ API Reference

The following two methods are missing from the C++ API Reference:

- **DDS_PublicationBuiltinTopicData::initialize_from(const DDS_DomainParticipantQos &dp_qos)**
- **DDS_SubscriptionBuiltinTopicData::initialize_from(const DDS_DomainParticipantQos &dp_qos)**

Please refer to the Discovery module in the C API Reference for a description of these functions.

[RTI Issue ID MICRO-7112]

9.5.3 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]

9.5.4 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]

9.5.5 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]

9.5.6 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]

9.5.7 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]

9.5.8 NaN and INF float and doubles are not detected and will not cause errors

Normally, *Connext Micro* discards samples with values that are out of range during serialization and de-serialization; however, Not a Number (NaN) and Infinite (INF) floating point and doubles are not detected and will not cause serialization or de-serialization errors.

[RTI Issue ID MICRO-5960]

9.5.9 Incorrect handling of RTPS messages with submessages from different participants

When an RTPS message that contains submessages from multiple participants is received, *Connext Micro* incorrectly treats each submessage as though it is from the participant whose GUID prefix is in the RTPS header. *Connext Micro* does not send RTPS messages with submessages from different participants, but other DDS vendors may do this, which leads to various communication issues and a lack of interoperability.

[RTI Issue ID MICRO-5984]

9.5.10 Ungracefully terminated QNX processes using SHMEM transport prevents 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 wait forever for that semaphore to be released.

Workaround: to enable new applications to start, RTI recommends stopping all applications, then cleaning up the Inter-Process Communication (IPC) resources before starting new applications.

[RTI Issue ID MICRO-6013]

9.5.11 Flow Controllers require RTOS

Flow controllers require an RTOS.

[RTI Issue ID MICRO-6648]

9.5.12 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]

9.5.13 Porting Guide does not include information about shared memory support

The *RTI Connext Micro* porting guide does not include information about porting the shared memory support. If the target does not support the POSIX shared memory API, it is necessary to compile *RTI Connext Micro* with *-DOSAPI_ENABLE_SHMEM=0*.

[RTI Issue ID MICRO-6650]

Chapter 10

Benchmarks

Performance benchmarks are no longer included with an *RTI Connext Micro* installation. Please refer to the RTI Connext Performance Benchmarks on RTI Community for more information.

Note: The *RTI Connext Performance Benchmarks* contain metrics for multiple products and versions, so please ensure that you refer to the appropriate section.

Chapter 11

Copyrights

Trademarks

Copy and Use Restrictions

Third-Party Software

Notices

Deprecations and Removals

Any deprecations or removals noted in this document serve as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software.

Deprecated means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in a release, RTI hereby provides customer notice that RTI reserves the right after one year from the date of such release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

Technical Support
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

Chapter 12

Third-Party 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.

12.1 Connext Micro Libraries

12.1.1 fnmatch

- Related to: Content-filtered topics, query conditions, partitions, multicast address management, topic filter in XML QoS profile.
- Software is included in the core middleware libraries.
- 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:
```

(continues on next page)

(continued from previous page)

12.1.2 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.
- License

(continues on next page)

(continued from previous page)

```
L. Peter Deutsch
ghost@aladdin.com
```

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

12.2.1 ANTLR

- This software is distributed with rtiddsgen (RTI Code Generator) as a jar file. The source code is not modified or shipped. In addition, the output produced by this software from a grammar file is part of the rtiddsgen JAR file. This has a dependency on ANTLR Runtime, StringTemplates v.3.2.1 and ST4 v.4.0.4.
- Version: Release 3.5.2
- License: https://www.antlr3.org/license.html

12.2.2 Apache Commons Lang

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

12.2.3 Apache Log4j 2

- This software is distributed with rtiddsgen (RTI Code Generator) as a jar file. The source code is not modified or shipped.
- Source: https://logging.apache.org/log4j/2.12.x/license.html
- Version: 2.17.1

12.2.4 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
- License: The Apache Software License, Version 2.0
  http://velocity.apache.org/engine/devel/license.html

12.2.5 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
- 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.
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
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.
```

12.2.6 Gson

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

12.3 Micro Application Generator (rtiddsmag)

12.3.1 Apache Commons CLI

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

12.3.2 Apache Commons Lang

- Used in Micro Application Generator. We only use the class StringUtils.
- Version 3.7
- License: Apache License Version 2.0 (full text found in the Appendix).

12.3.3 Apache Log4j 2

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

12.3.4 Apache Velocity

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

12.3.5 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
- 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.
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
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
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.
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

12.3.6 Extended StAX API

- Version 1.8
- License: https://www.eclipse.org/org/documents/edl-v10.php

12.3.7 Fast Infoset

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

12.3.8 Istack Common Utility Code Runtime

- Version 3.0.7
- License: https://javaee.github.io/glassfish/LICENSE

12.3.9 JavaBeans Activation Framework API

- Version 1.2.0
- License: https://oss.oracle.com/licenses/CDDL+GPL-1.1

12.3.10 Javax Annotation API

- Version 1.3.2
- License: https://github.com/javaee/javax.annotation/blob/master/LICENSE

12.3.11 JAXB API

- Version 2.3.1
- License: https://javaee.github.io/jaxb-v2/LICENSE

12.3.12 JAXB Runtime

- Version 2.3.1
- License: https://javaee.github.io/jaxb-v2/LICENSE

12.3.13 Simple Logging Facade for Java (SLF4J)

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

(continues on next page)

(continued from previous page)

```
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE,  ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

12.3.14 TXW2

- Version 2.3.1
- License: https://oss.oracle.com/licenses/CDDL+GPL-1.1

12.4 Appendix – Open Source Software Licenses

12.4.1 Apache License version 2.0, January 2004 (http://www.apache.org/licenses/)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

12.4.2 GNU GENERAL PUBLIC LICENSE Version 2, June 1991

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

Chapter 13

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 14

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.