RTI Connext Micro

Release Notes

Version 2.4.1

Your systems. Working as one.

Technical Support
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089

Phone:	(408) 990-7444
Email:	support@rti.com
Website:	http://www.rti.com

Contents

Release Notes for RTI Connext Micro

This document provides release-specific information and identifies highlights and updates in *RTI® Connext™ Micro* version 2.4.1.

1 Supported Operating Systems and Compilers

This section describes the platforms supported by *RTI Connext Micro* 2.4.1.

Supported Platforms:

- ❏ Linux (Intel®)
- ❏ Windows®
- ❏ VxWorks®
- ❏ Android (NDK)

The following tables provide specific details per supported platform.

Table 1.1 **Linux Supported Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Red Hat Enterprise Linux 6.0, 6.1 (2.6 kernel)	x86	gcc 4.4.5	i86Linux2.6gcc4.4.5

Table 1.2 **Linux Build Instructions**

RTI Architecture	Required System Libraries	Required Compiler Flags
i86Linux2.6gcc4.4.5 (Release)	-ldl -lnsl -lm- lpthread -lrt	-fPIC -DLINUX -DRTI_GCC4 -DRTI_LINUX26 -DRTI_LINUX -DRTI_POSIX_THREADS -DRTI_POSIX_SEMAPHORES -DRTI_CPU_AFFINITY -O -Wall -Wno-unknown-pragmas -DRTI_UNIX -DRTS_UNIX -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DRTI_ENDIAN_LITTLE -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6 -DNDEBUG
i86Linux2.6gcc4.4.5 (Debug)	-ldl -lnsl -lm- lpthread -lrt	-fPIC -DLINUX -DRTI_GCC4 -DRTI_LINUX26 -DRTI_LINUX -DRTI_POSIX_THREADS -DRTI_POSIX_SEMAPHORES -DRTI_CPU_AFFINITY -g -DRTI_PRECONDITION_TEST -Wall -Wno-unknown-pragmas -DRTI_UNIX -DRTS_UNIX -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DRTI_ENDIAN_LITTLE -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6

Table 1.3 **Windows Supported Platforms**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Windows 7	x86	Visual Studio 2010	i86Win32VS2010

Table 1.4 **Windows Build Instructions**

RTI Architecture	Required System Libraries	Required Compiler Flags
i86Win32VS2010	netapi32.lib advapi32.lib user32.lib WS2_32.lib Iphlpapi.lib winmm.lib	/D “RTI_WIN32” /D “WIN32”/Gd /MT (Static Release) /MTd (Static Debug) /MD (Dynamic Release) /MDd (Dynamic Debug) /D “NDDS_DLL_VARIABLE (Dynamic) /D “NDEBUG” (Release)

Table 1.5 **VxWorks Supported Platforms**[1]

Operating System	CPU	Compiler, Required System Libraries	RTI Architecture Abbreviation
VxWorks 6.9	Pentium 32-bit	gcc 4.3.3	For Kernel Modules: pentiumVx6.9gcc4.3.3 For Real Time Processes: pentiumVx6.9gcc4.3.3_rtp
	PPC32	gcc 4.3.3	For Kernel Modules: ppc604Vx6.9gcc4.3.3 For Real Time Processes: ppc604Vx6.9gcc4.3.3_rtp
VxWorks Cert 6.6.4.1	PPC32 e500v2	gcc 4.1.2	ppce500v2VxCert6.6.4.1gcc4.1.2

1. For use with host platforms (e.g., Windows or Linux) as supported by Wind River Systems

Table 1.6 **VxWorks Build Instructions**

RTI Architecture	**Required Compiler Flags**
pentiumVx6.9gcc4.3.3 (Release)	-march=pentium -m32 -fno-builtin -ansi -DCPU=PENTIUM -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_GCC4 -D_WRS_KERNEL -D__PROTOTYPE_5_0 -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -O -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PENTIUM -DRTI_ENDIAN_LITTLE -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6 -DNDEBUG
pentiumVx6.9gcc4.3.3 (Debug)	-march=pentium -m32 -fno-builtin -ansi -DCPU=PENTIUM -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_GCC4 -D_WRS_KERNEL -D__PROTOTYPE_5_0 -g -DRTI_PRECONDITION_TEST -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PENTIUM -DRTI_ENDIAN_LITTLE -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6

Table 1.6 **VxWorks Build Instructions**

RTI Architecture	**Required Compiler Flags**
pentiumVx6.9gcc4.3.3_rtp (Release)	-march=pentium -m32 -ansi -DCPU=PENTIUM -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_GCC4 -mrtp -D__PROTOTYPE_5_0 -O -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DRTI_RTP -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PENTIUM -DRTI_ENDIAN_LITTLE -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6 -DNDEBUG
pentiumVx6.9gcc4.3.3_rtp (Debug)	-march=pentium -m32 -ansi -DCPU=PENTIUM -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_GCC4 -mrtp -fPIC -D__PROTOTYPE_5_0 -g -DRTI_PRECONDITION_TEST -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DRTI_RTP -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PENTIUM -DRTI_ENDIAN_LITTLE -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6

Table 1.6 **VxWorks Build Instructions**

RTI Architecture	Required Compiler Flags
ppc604Vx6.9gcc4.3.3 (Release)	-m32 -mstrict-align -ansi -fno-builtin -mlongcall -DCPU=PPC32 -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_USE_MUNCH -DRTI_GCC4 -D_WRS_KERNEL -D__PROTOTYPE_5_0 -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -O2 -fno-strict-aliasing -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PPC32 -DRTI_ENDIAN_BIG -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6 -DNDEBUG
ppc604Vx6.9gcc4.3.3 (Debug)	-m32 -mstrict-align -ansi -fno-builtin -mlongcall -DCPU=PPC32 -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_USE_MUNCH -DRTI_GCC4 -D_WRS_KERNEL -D__PROTOTYPE_5_0 -g -DRTI_PRECONDITION_TEST -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PPC32 -DRTI_ENDIAN_BIG -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6

Table 1.6 **VxWorks Build Instructions**

RTI Architecture	Required Compiler Flags
ppc604Vx6.9gcc4.3.3_rtp (Release)	-mhard-float -mstrict-align -m32 -mregnames -ansi -mlongcall -DCPU=PPC32 -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_USE_MUNCH -DRTI_GCC4 -DRTI_RTP -mrtp -D__PROTOTYPE_5_0 -O2 -fno-strict-aliasing -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DRTI_RTP -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PPC32 -DRTI_ENDIAN_BIG -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6 -DNDEBUG
ppc604Vx6.9gcc4.3.3_rtp (Debug)	-mhard-float -mstrict-align -m32 -mregnames -ansi -mlongcall -DCPU=PPC32 -DTOOL_FAMILY=gnu -DTOOL=gnu -DRTI_USE_MUNCH -DRTI_GCC4 -DRTI_RTP -mrtp -fPIC -D__PROTOTYPE_5_0 -g -DRTI_PRECONDITION_TEST -Wall -Wno-unknown-pragmas -DRTI_VXWORKS -DRTS_VXWORKS -DRTI_RTP -DVXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=9 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=PPC32 -DRTI_ENDIAN_BIG -DRTI_THREADS -DRTI_MULTICAST -DRTI_SHARED_MEMORY -DRTI_IPV6

Table 1.6 **VxWorks Build Instructions**

RTI Architecture	Required Compiler Flags
ppce500v2VxCert6.6.4.1gcc4.1.2 (Release)	-m32 -mstrict-align -mcpu=8548 -mfloat-gprs=double -mspe=yes -mabi=spe -ansi -fno-exceptions -fno-builtin -Wall -msdata=eabi -DCPU=PPC32 -DTOOL_FAMILY=gnu -DTOOL=e500v2gnu -VXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=6 -DCERT -DRTI_CERT -DRTI_ENDIAN_BIG -DRTI_VXWORKS -DOSAPI_PLATFORM=4 -O -DNDEBUG
ppce500v2VxCert6.6.4.1gcc4.1.2 (Debug)	-m32 -mstrict-align -mcpu=8548 -mfloat-gprs=double -mspe=yes -mabi=spe -ansi -fno-exceptions -fno-builtin -Wall -msdata=eabi -DCPU=PPC32 -DTOOL_FAMILY=gnu -DTOOL=e500v2gnu -VXWORKS_MAJOR_VERSION=6 -DVXWORKS_MINOR_VERSION=6 -DCERT -DRTI_CERT -DRTI_ENDIAN_BIG -DRTI_VXWORKS -DOSAPI_PLATFORM=4 -O -g

Table 1.7 **Android Supported Platform**

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Android NDK, Revision 8d	ARMv7	gcc 4.7	armv7leAndroidR8Dgcc4.7

Table 1.8 **Android Build Instructions**

RTI Architecture	Required Compiler Flags
armv7leAndroidR8Dgcc4.7	-march=armv7-a -mfpu=neon -msoft-float -mlong-calls

To be able to use the *RTI Connext Micro* libraries, the Android manifest should have the following permissions:

- android.permission.INTERNET

- ❑ android.permission.CHANGE_WIFI_MULTICAST_STATE
- ❑ android.permission.CHANGE_WIFI_STATE

2 Installing RTI Connext Micro

RTI Connext Micro is available for installation in separate host and target packages.

You will have *at least* two bundles (as **.zip** or **.tar.gz** archives): one for your host platform and at least one target platform. The host package is platform agnostic; it provides documentation, header files, examples, and the *rtiddsgen* utility. The target package provides platform-specific libraries and utilities. Expand them into the same directory of your choice, as described below.

2.1 Installing on a Windows System

The distribution is packaged in one or more **.zip** files. Unzip them into a directory of your choice.

2.2 Installing on a UNIX-Based System

The distribution is packaged in one or more files. Unpack them as described below. You do not need to be logged in as root during installation.

1. Make sure you have GNU's version of **unzip** and **tar** (which handles long file names). On Linux systems, this is the default **tar** executable.
2. Create a directory for *RTI Connext Micro*.
3. Move the downloaded file(s) into your newly created directory.
4. Extract the distribution from the uncompressed files. For example:

```
unzip RTI_Connext_Micro-Host-2.4.1.zip
tar zxvf RTI_Connext_Micro_Target-2.4.1-i86Linux2.6gcc4.4.5.tar.gz
```

Assuming your directory is **/home/user/**, you will end up with **/home/user/rti_connext_micro.2.4.1.**

2.3 Unpacking Buildable Source

For the optional buildable source bundle, unpack it in the same directory (from the example above, **/home/user**).

3 RTI Connext Micro Features

Important features of *RTI Connext Micro* are described here.

3.1 Modular Architecture

The flexible architecture of *RTI Connext Micro* 2.4.1 enables users to create, change, and connect the modules that make up *RTI Connext Micro*'s middleware and network stack.

Each module within this architecture has the same data and control interface. For example, the modules for DDS, RTPS, and the UDPv4 transport all implement the same interfaces, even though they each occupy a different level in the middleware stack. Having the same interface for each module enables easier extensibility and testability.

Users can create and add a custom transport to the stack as a new module. Users can also remove or revise existing modules. For example, it is possible to remove DDS and send/receive data over pure RTPS.

3.2 Buildable Source Code

Users can build *RTI Connext Micro* from the available buildable source code bundle. This enables users to port *RTI Connext Micro* to new platforms, without having to wait for an official release. With the open-source build tool, *cmake* (ww.cmake.org), users can generate native makefiles and workspaces for any of the *cmake*-supported platforms. Note that the shipped **CMakeLists.txt** file is set up to generate a Linux makefile and a Windows project.

A README within the buildable source directory provides instructions on using *cmake* to build *RTI Connext Micro*. Consult *cmake* documentation or contact RTI Support to create the cross-compilation toolchain files necessary for VxWorks and other embedded platforms.

To request the separate buildable source code bundle, please contact **sales@rti.com**.

3.2.1 Build Configuration

The top-level configuration file, **rti_me_config.h**, has all the build configuration flags and settings that a user building from buildable source may need to set.

3.2.2 Porting to New Platforms

RTI Connext Micro has an operating system abstraction layer (OSAPI) to support running on different platforms. OSAPI supports functionality typically provided by operating system calls, devices drivers, and the standard C library.

Consult the OSAPI API in the HTML documentation when porting to a new platform, as most source code changes will likely be within OSAPI.

3.3 Type-Support Code Generation

RTI Connext Micro provides the *rtiddsgen* utility for generating type support code that is necessary to publish and subscribe user-defined data types.

The script to run *rtiddsgen* is in **rti_connext_micro.2.4.1/rtiddsgen/scripts**.

An example command for generating code for a type Foo.idl:

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

In this release, *rtiddsgen* supports generating code from IDL 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

Unlike in *RTI Connext*, this version *rtiddsgen* does not generate example HelloWorld applications. Instead, users should refer to the examples shipped with *RTI Connext Micro*.

3.4 Static and Dynamic Endpoint Discovery

Discovery is the process by which DDS participants, writers, and readers determine whether communication should be established between one another. For a DDS endpoint (i.e., a writer or reader) to discover another entity, it must first receive some state (i.e., endpoint discovery information) about the other entity (e.g., reliability QoS, Topic name, etc.). Then it establishes communication only with other entities that have states compatible with its own state.

RTI Connext Micro provides two mechanisms for endpoint discovery between writers and readers: static endpoint discovery is the manual registration of remote endpoint information by the user; dynamic endpoint discovery is the automatic exchange of endpoint information over a reliable channel, and is transparent to the user.

Example code for static and dynamic discovery is provided in the HelloWorld_dpse and HelloWorld_dpde examples, respectively.

3.5 User Documentation

3.5.1 API Reference Documentation

The complete *RTI Connext Micro* API is accessible through hyperlinked HTML documentation.

For porting *RTI Connext Micro* to a new platform, the HTML documentation includes the *RTI Connext Micro Porting Guide* module, which describes the features, APIs, and configuration that need to be considered.

3.5.2 DDS API Guide

As a small-footprint implementation of DDS, *RTI Connext Micro* supports a subset of the standard DDS API and Quality of Service (QoS) policies.

The API and QoS supported in this version are described in the accompanying *API Guide* (**RTI_Connext_Micro_APIGuide.pdf**).

3.6 Code Examples

The provided example applications will help users become familiar with *RTI Connext Micro*. Each example has a README with instructions on how to build and run an application.

All examples are available in C, while the HelloWorld_dpde example is available in C++.

HELLOWORLD_DPSE EXAMPLE Shows how to use *rtiddsgen* to generate type-support code from a simple HelloWorld IDL-defined type. This example creates a publisher and subscriber, and uses dynamic participant, static endpoint discovery to establish communication.

HELLOWORLD_DPDE EXAMPLE Same as the HelloWorld_dpse example, except it uses dynamic participant, *dynamic* endpoint discovery. This example is available in both C and C++.

HELLOWORLD_DPDE_WAITSET Same as the HelloWorld_dpde example, except it uses waitsets instead of listener callbacks to access received data.

HELLOWORLD_ANDROID Example application using Android NDK.

HELLOWORLD_STATIC_UDP Example using static configuration of network interfaces.

RTPS Example of an RTPS emitter that bypasses the DDS module and APIs to send RTPS discovery and user data.

LATENCY Measures the end-to-end latency of *RTI Connext Micro*.

THROUGHPUT Measures the end-to-end throughput of *RTI Connext Micro*.

3.7 Interoperability and Compatibility

RTI Connext Micro supports a subset of the submessages defined by the Real-Time Publish-Subscribe (RTPS) interoperability specification: DATA, ACKNACK, HEARTBEAT, and GAP submessages are supported, as well as INFO_TS and INFO_DST. Note that fragmented data submessages are not supported. The messages are compatible with Wireshark and its RTPS packet dissector.

Interoperability between 2.4.1 and both *RTI Data Distribution Service* 4.5d and *RTI Connext* 5.1.0 has been verified between example HelloWorld applications and with the *rtiddsspy* utility.

Compatibility with the following RTI tools and services has not been verified: *Routing Service*, *Federation Service*, *Analyzer*, *Spreadsheet Add-In*, *Real-Time Connect*, and *Recording Service*. Nevertheless, they may currently be compatible, assuming that the *RTI Connext Micro* application is using the dynamic endpoint discovery plugin. *Monitor* and *Limited Bandwidth Plug-Ins* are not compatible with *RTI Connext Micro* 2.4.1.

4 New Features in 2.4.1

This section describes new features added since 2.3.2.

4.1 New -micro option for rtiddsgen

The *rtiddsgen* version shipped with this release of *RTI Connext Micro* introduces a new command line argument, "-micro", which enables users to explicitly target *RTI Connext Micro* when using the code generation tool.

[RTI Issue ID MICRO-779]

Previously this was implicitly implied by the use of values "microC" or "microC++" for command line argument "-language".

This prevented *rtiddsgen* to be able to produce an help output specific for *RTI Connext Micro*. This functionality is now available to users, who may use arguments "-micro -help" to visualize a list of all command line arguments supported by *rtiddsgen* when targeting RTI Connext Micro.

4.2 New language options for generating code for RTI Connext Micro using rtiddsgen

The new version of *rtiddsgen* that is shipped with *RTI Connext Micro* introduces new values for the "-language" command line argument, which should be adopted by users instead of the previously supported "microC" and "microC++".

In order to generated custom data-types support code that can be used with *RTI Connext Micro*, user shall specify the following command line options.

❏ Target language C

```
-micro -language C
```

❏ Target language C++

```
-micro -language C++
```

[RTI Issue ID MICRO-79]

4.3 CMake Configuration for Buildable Source

The provided CMake configuration for building *RTI Connext Micro* from source code has been updated:

- ❏ Configuration options have been simplified, including selecting a target operating system, and for including C++ API support
- ❏ Output libraries are now identical to pre-built shipped libraries of *RTI Connext Micro*. The previous configuration instead produced a single library.

[RTI Issue ID MICRO-112, MICRO-770, MICRO-805]

4.4 DDS_WaitSet and DDS_Condition APIs For C++

This release officially introduces full support of DDS_WaitSet and DDS_Condition APIs also for C++ users of *RTI Connext Micro*.

The following operations are officially supported:

- ❏ DDSWaitSet::DDSWaitSet()
- ❏ DDSWaitSet::wait()
- ❏ DDSWaitSet::attach_condition()
- ❏ DDSWaitSet::detach_condition()
- ❏ DDSWaitSet::get_conditions()
- ❏ DDSCondition::get_trigger_value()
- ❏ DDSGuardCondition::DDSGuardCondition()
- ❏ DDSGuardCondition::set_trigger_value()
- ❏ DDSStatusCondition::set_enabled_statuses()
- ❏ DDSStatusCondition::get_enabled_statuses()
- ❏ DDSStatusCondition::get_entity()

[RTI Issue ID MICRO-685]

4.5 New HelloWorld_dpde_waitset Example

A new code example, HelloWorld_dpde_waitset, has been added to the set of examples shipped with *RTI Connext Micro*.

This example is based on the preexisting HelloWorld_dpde example and it exemplifies the use of the recently introduced DDS_WaitSet and DDS_Condition APIs.

By leveraging a few pre-processor macros, the example also provides an equivalent implementation of its behavior using traditional listeners on DDS entities. Users may switch between one implementation and the other by modifying values of these macro and by recompiling the example's source code.

[RTI Issue ID MICRO-708]

4.6 Access to Status of DDS Entities by C++ APIs

Operations of the DDS API which grant users access to data structures describing the current internal information about a particular entity status have now been fully implemented and they are officially supported by *RTI Connext Micro's* C++ API.

The following methods have been introduced:

- ❏ DDSDataReader::get_sample_rejected_status
- ❏ DDSDataReader::get_sample_lost_status
- ❏ DDSDataReader::get_subscription_matched_status
- ❏ DDSDataReader::get_requested_incompatible_qos_status
- ❏ DDSDataReader::get_requested_deadline_missed_status
- ❏ DDSDataReader::get_instance_replaced_status
- ❏ DDSDataReader::get_liveliness_changed_status
- ❏ DDSDataWriter::get_publication_matched_status
- ❏ DDSDataWriter::get_offered_incompatible_qos_status
- ❏ DDSDataWriter::get_offered_deadline_missed_status
- ❏ DDSDataWriter::get_liveliness_lost_status
- ❏ DDSTopic::get_inconsistent_topic_status

[RTI Issue ID MICRO-784]

4.7 Improved support for listeners of DDS entities in C++ API

Support for operations allowing the setting and retrieval of listener objects from DDS entities has been extended and it is now fully available to users of *RTI Connext Micro's* C++ API.

The following methods have been implemented and they are now fully supported as part of the C++ API:

- ❏ DDSDomainParticipant::set_listener
- ❏ DDSDomainParticipant::get_listener
- ❏ DDSSubscriber::set_listener
- ❏ DDSSubscriber::get_listener

- ❏ DDSPublisher::set_listener
- ❏ DDSPublisher::get_listener
- ❏ DDSTopic::set_listener
- ❏ DDSTopic::get_listener
- ❏ DDSDataWriter::set_listener
- ❏ DDSDataReader::set_listener

[RTI Issue ID MICRO-795]

4.8 New Operations FooTypeSupport_register/unregister_type for User Data-Types

The new version of *rtiddsgen* that is shipped with *RTI Connext Micro* now generates two new additional operation which can be used by users to more easily register a data-type on a DDS_DomainParticipant. In particular, these operation allow the specification of an empty (null) type name at registration time. The default name associated with the type will be used in this case.

The new operations are generated only for top-level data-types which are the only ones usable for the creation of a DDS_Topic.

[RTI Issue ID MICRO-796]

4.9 Sample Info Provided With DataReaderListener's Sample Filter Callback

DataReaderListener.on_before_sample_commit() now provides sample info of the sample being filtered in a new parameter, sample_info.

[RTI Issue ID MICRO-809]

4.10 More Information Provided in DDS_SampleLostStatus Type

The DDS_SampleLostStatus type has two new fields:

- ❏ kind
- ❏ sample_info

The kind (of type DDS_SampleLostStatusKind) indicates why a sample was lost. For samples lost for any reason other than DDS_SAMPLE_LOST_BY_DATAWRITER, sample_info gives additional information about the sample.

[RTI Issue ID MICRO-919]

5 What’s Fixed in 2.4.1

This section describes bugs fixed in this release since 2.3.2.

5.1 Full Support of -namespace Option for rtiddsgen

The new version of rtiddsgen shipped with *RTI Connext Micro* introduces official support for the "-namespace" command line argument when the tool is used for the generation of C++ type support code.

If the option is specified on the command line, IDL modules will be translated into C++ namespaces instead of being used as prefixes to the name of data-types defined within them.

[RTI Issue ID MICRO-153]

5.2 Redundant Code Not Generated for Non Top-Level Types

Previous version of *rtiddsgen* that were shipped with *RTI Connext Micro* included, in the generated code for user-specified data-types, several support operations which were not required by data-types which cannot be used as top-level types for a DDS_Topic.

These redundant operation have now been removed from the generation outcome and they are only produced for top-level types.

[RTI Issue ID MICRO-678]

5.3 Setting Real-Time Clock for VxWorks Platform

For VxWorks platforms, when initializing an application, *RTI Connext Micro* set the real-time clock (i.e. clock_settime) to a predefined value. This could have interfered with user configured time, thus this release no longer sets the real-time clock.

[RTI Issue ID MICRO-692]

5.4 IDL Copy Directives Now Supported by rtiddsgen

Support for IDL copy directives (such as "//@copy", "//@copy-c", ...) has been improved and the following copy directives are now fully supported in user-provided IDL files:

- //@copy
- //@copy-declaration

- ❑ //@copy-c
- ❑ //@copy-c-declaration

[RTI Issue ID MICRO-740]

5.5 Incompatible Encoding of DDS_Duration_t with RTI Connext DDS

The DDS_Duration_t datatype was not correctly encoded by *RTI Connext Micro* when sent over the wire. Specifically, the nanoseconds field of Duration_t was not correctly encoded. This was not a problem between *RTI Connext Micro* applications, but when inter-operating with *RTI Connext DDS* applications, the durations of Liveliness lease_duration and Deadline period were misinterpreted and resulted in incorrect timing. A workaround was to multiply the nanoseconds field of the *RTI Connext Micro* application's durations by 4 when communication with *RTI Connext DDS*.

[RTI Issue ID MICRO-768]

5.6 Premature Wakeup from Semaphore Take for POSIX Platforms

A problem existed on platforms using POSIX threads (e.g. Linux) where taking a semaphore with a timeout may have woken up prematurely. This could have resulted premature wake-up for Waitsets.

[RTI Issue ID MICRO-772]

5.7 Failed Compilation of C++ Generated Code from rtiddsgen

A bug has been fixed in the code generation templates, used by *rtiddsgen* to produce support code used by the C++ API for user-defined custom data-types. The bug caused erroneous code to be generated for non top-level data-types in the resulting source files. This code prevented the generated source code from being successfully compiled.

These additional and unnecessary parts have now been removed.

[RTI Issue ID MICRO-777, MICRO-778]

5.8 DDS_Topic_get_inconsistent_topic_status in C API

The operation DDS_Topic_get_inconsistent_topic_status has been added to *RTI Connext Micro's* C API and it is now fully supported.

[RTI Issue ID MICRO-783]

5.9 DDS_StatusCondition's Trigger Value Not Updated When DDS_DATA_AVAILABLE_STATUS Is Reset

A bug has been fixed which prevented a DDS_StatusCondition instance, associated with a DDS_DataReader, DDS_Subscriber, or DDS_DomainParticipant, from correctly transitioning its trigger value from true to false when state DDS_DATA_AVAILABLE_STATUS was consumed and deactivated in the associated DDS entity.

This behavior caused the wait() operation of any DDS_WaitSet, that had the DDS_StatusCondition instance attached to it, to always immediately return successfully, after the DDS_StatusCondition's trigger value had been transitioned to active. Note that it was still possible for the condition's trigger value to transition to false state if another of the enabled statuses transitioned to false in the associated DDS entity, causing the DDS_StatusCondition to properly update its state according to the DDS entity's one.

[RTI Issue ID MICRO-786]

5.10 DDS_WaitSet_get_conditions Supported by C API

The operation DDS_WaitSet_get_conditions has now been fully implemented and it can be now used to access the list of DDS_Conditions attached to a DDS_WaitSet instance.

[RTI Issue ID MICRO-788]

5.11 Reliable, Keep-Last DataReader May Stop Receiving Samples

A reliable DataReader with KEEP_LAST history may have stopped receiving new samples, after previous samples were received out of order. This was due to a bug where there were no available resources of the DataReader to receive samples that would allow the out of order samples to be presented in order to the application.

[RTI Issue ID MICRO-790]

5.12 Memory Dynamically Allocated By DPSE After Initialization

RTI Connext Micro is designed not to allocate memory dynamically after an entity has been created and enabled. However, a bug in the implementation for Dynamic Participant, Static Endpoint (DPSE) discovery resulted in the dynamic allocation of memory during participant discovery. The memory allocated was to increase the maximum size of sequences for storing received participant discovery data. This release has been fixed to do this allocation during initialization.

[RTI Issue ID MICRO-803]

5.13 Application Crashed When Looking Up DomainParticipant

Previously, an application would crash when DDS_DomainParticipantFactory_lookup_participant() was called before the first invocation of DDS_DomainParticipantFactory_get_instance(). This was caused by improper initialization of the DomainParticipantFactory and has been fixed in this release.

[RTI Issue ID MICRO-819]

5.14 Setting Log Verbosity May Have Been Ineffective

Logging verbosity is configured by calling OSAPI_Log_set_verbosity(). It should be configurable at any time. However, a bug existed where the first call to DDS_DomainParticipantFactory_get_instance() reset the log verbosity to OSAPI_LOGKIND_ERROR. This caused calls to OSAPI_Log_set_verbosity() for verbosity not equal OSAPI_LOGKIND_ERROR to be ineffective when called before the first DDS_DomainParticipantFactory_get_instance().

[RTI Issue ID MICRO-820]

5.15 Waitset's Wait may have Incorrectly Timed Out when Listener also Installed

Given a DataReader and a DDS status handled by both the DataReaderListener and a Waitset, the Waitset's wait operation may not have correctly woken up, resulting in a timeout. For example, having DDS_DATA_AVAILABLE_STATUS handled by both the listener's on_data_available callback and as an enabled status of a StatusCondition attached to a Waitset could have resulted in the Waitset waiting to timeout even though data was actually available. As a workaround, you may have disabled the listener and instead relied only on the Waitset. This workaround is no longer needed.

[RTI Issue ID MICRO-969]

5.16 Data not Received due to Samples not being Sent by all Transports

RTI Connext Micro supports two built-in transports: UDP and Intra, where Intra only transports data between endpoints belonging to the same DomainParticipant. With both transports enabled (which is the default setting), a bug introduced in Connext Micro 2.3.1 caused written samples to be sent over only one transport. This could result in samples not being received by a remote DomainParticipant, where samples are sent over Intra but not over UDP.

[RTI Issue ID MICRO-866]

5.17 Static Endpoint Discovery Restrictively Required Unique Object IDs Across All Remote Endpoints

When using static endpoint discovery (DPSE), *RTI Connext Micro* incorrectly enforced a requirement that the object_id for statically asserted remote endpoints must be unique across all remote endpoints, as opposed to just between remote endpoints within the same participant. This restriction has been removed.

[RTI Issue ID MICRO-211]

5.18 Possible Non-Discovery or Non-Communication between Connext Micro using DPSE and a Non-Micro Application

The participant discovery messages sent by a Connext Micro application using the Dynamic-Participant Static-Endpoint (DPSE) discovery plugin contained an incorrect value for the Built-in Endpoint Mask parameter. The bits of the mask indicate which built-in discovery endpoints are enabled, and for DPSE the bits corresponding to the participant built-in writer and reader were not set. This could cause other DDS implementations not to discover, communicate, or correctly update statuses with Connext Micro using DPSE. This problem has been fixed in this release.

[RTI Issue ID MICRO-974]

6 Known Issues

6.1 Incomplete Documentation for non-DDS APIs

RTI Connext Micro does not yet provide documentation for all non-DDS modules. The internal modules missing documentation include the run-time (RT), the writer and reader history queues (wh_sm, rh_sm), RTPS, and NETIO.

6.2 C++ APIs Not Supported for Android Platform

The Android platform (armv7leAndroidR8Dgcc4.7) does not support C++ APIs in this release.

6.3 Waitset's Wait May Return Same Condition More Than Once For a Single Triggering Event

When a Condition attached to a Waitset is triggered, it should be returned by a following call to that Waitset's wait(), as long as it is not reset. However, due to a race condition between a Waitset and its attached Conditions, the same Condition may be returned by two successive calls to wait(). Even though the Condition was unnecessarily returned again, if inspected the Condition's trigger value would be correctly seen as inactive.

6.4 Waitset Delete Not Safe For Concurrent Use

A Waitset's delete() operation is not safe for concurrent use when other threads may be calling any operations of the Waitset. In general, access to a WaitSet should be guaranteed to have ceased before its delete() operation is called.