RTI Connext DDS Core Libraries

Platform Notes

Version 6.1.2

© 2026 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
August 2026.

Trademarks

RTI, Real-Time Innovations, Connext, NDDS, the RTI logo, 1RTI and the phrase, "Your Systems. Working as one." are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished solely under and subject to RTI's standard terms and conditions available at https://www.rti.com/terms and in accordance with your License Acknowledgement Certificate (LAC) and Maintenance and Support Certificate (MSC), except to the extent otherwise accepted in writing by a corporate officer of RTI.

This is an independent publication and is neither affiliated with, nor authorized, sponsored, or approved by, Microsoft Corporation.

The security features of this product include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Notice

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/

Contents

Chapter 1 Supported Platforms

1.1 Paths Mentioned in Documentation 4

Chapter 2 AIX Platforms

2.1 Multicast Support 9

2.2 Transports 9

2.2.1 Notes for Using Shared Memory 9

2.3 Unsupported Features 9

2.4 Thread Configuration 10

2.4.1 Changing Thread Priority 11

2.5 Libraries Required for Using Distributed Logger 12

2.6 Libraries Required for Using Monitoring 12

2.7 Libraries Required for Zero Copy Transfer Over Shared Memory 12

Chapter 3 Android Platforms

3.1 Support for Modern C++ API 18

3.2 Multicast Support 19

3.3 Transports 19

3.4 Unsupported Features 19

3.5 Monotonic Clock Support 20

3.6 Thread Configuration 20

3.7 Support for Remote Procedure Calls (RPC) 21

3.8 Libraries Required for Using Distributed Logger 21

3.9 Libraries Required for Using Monitoring 22

3.10 Libraries Required for Using RTI Real-Time WAN Transport APIs 22

3.11 Libraries Required for Using RTI Secure WAN Transport APIs 23

3.12 Libraries Required for Using RTI TCP Transport and TLS Support APIs 23

Chapter 4 INTEGRITY Platforms

4.1 Required Patch for INTEGRITY 11.0.4 26
4.2 Support for Modern C++ API 28
4.3 Multicast Support 29
4.4 Supported Transports 29
4.5 Unsupported Features 29
4.6 Thread Configuration 29
4.6.1 Socket-Enabled and POSIX-Enabled Threads are Required 31
4.7 Libraries Required for Using Distributed Logger 31
4.8 Libraries Required for Using Monitoring 32
4.9 Libraries Required for Using RTI Real-Time WAN Transport APIs 32
4.10 Libraries Required for Zero Copy Transfer Over Shared Memory 33
4.11 Running over IP Backplane on a Dy4 Champ-AVII Board 33
4.12 Out-of-the-box Transport Compatibility with Other Connext DDS Platforms 34
4.12.1 Smaller Shared-Memory Receive-Resource Queue Size 34
4.12.2 Using Shared Memory on INTEGRITY Systems 35
4.12.3 Shared Memory Limitations on INTEGRITY Systems 35
4.13 Using rtiddsping and rtiddsspy on INTEGRITY Systems 36
4.14 Issues with INTEGRITY Systems 36
4.14.1 Delay When Writing to Unreachable Peers 36
4.14.2 Linking with ‘libivfs.a’ without a File System 36
4.14.3 Compiler Warnings Regarding Unrecognized #pragma Directives 36
Chapter 5 Linux Platforms
5.1 Support for Modern C++ API 46
5.2 Multicast Support 47
5.3 Supported Transports 47
5.3.1 Shared Memory Support 47
5.4 Unsupported Features 48
5.5 Limitations of FACE Architectures 48
5.6 Monotonic Clock Support 49
5.7 Thread Configuration 49
5.7.1 Support for Controlling CPU Core Affinity for RTI Threads 49
5.7.2 Using REALTIME_PRIORITY 51
5.8 Durable Writer History and Durable Reader State Features 51
5.9 Support for 'Find Package' CMake Script 51
5.10 Backtrace Support 51
5.11 Support for Remote Procedure Calls (RPC) 52

5.12 Libraries Required for Using Distributed Logger 52
5.13 Libraries Required for Using Monitoring 53
5.14 Libraries Required for Using RTI Real-Time WAN Transport APIs 54
5.15 Libraries Required for Using RTI Secure WAN Transport APIs 54
5.16 Libraries Required for Using RTI TCP Transport and TLS Support APIs 55
5.17 Libraries Required for Zero Copy Transfer Over Shared Memory 56
Chapter 6 macOS Platforms
6.1 Installation Notes for macOS 11 Systems on Arm v8 CPUs 62
6.2 Support for Modern C++ API 62
6.3 Multicast Support 62
6.4 Supported Transports 62
6.5 Unsupported Features 63
6.6 System Integrity Protection (SIP) 63
6.6.1 SIP and Java Applications 63
6.6.2 SIP and Connext Tools, Infrastructure Services, and Utilities 64
6.7 Thread Configuration 64
6.8 Support for 'Find Package' CMake Script 66
6.9 Backtrace Support 66
6.10 Resolving NDDSUtility_sleep() Issues 66
6.11 Support for Remote Procedure Calls (RPC) 67
6.12 Libraries Required for Using Distributed Logger 67
6.13 Libraries Required for Using Monitoring 68
6.14 Libraries Required for Using RTI Real-Time WAN Transport APIs 68
6.15 Libraries Required for Using RTI Secure WAN Transport APIs 69
6.16 Libraries Required for Using RTI TCP Transport APIs 69
6.17 Libraries Required for Zero Copy Transfer Over Shared Memory 70
Chapter 7 QNX Platforms
7.1 Required Change for Building with C++ Libraries for QNX Platforms 80
7.2 Support for Modern C++ API 80
7.3 Multicast Support 81
7.4 Supported Transports 81
7.5 Unsupported Features 82
7.6 Monotonic Clock Support 82
7.7 Thread Configuration 82
7.7.1 Support for Controlling CPU Core Affinity for RTI Threads 84
7.8 Support for Remote Procedure Calls (RPC) 84

7.9 Libraries Required for Using Distributed Logger 84
7.10 Libraries Required for Using Monitoring 84
7.11 Libraries Required for Using RTI Real-Time WAN Transport APIs 85
7.12 Libraries Required for Using RTI Secure WAN Transport APIs 86
7.13 Libraries Required for Using RTI TCP Transport APIs and TLS Support 86
7.14 Libraries Required for Zero Copy Transfer Over Shared Memory 87
7.15 Restarting Applications on QNX Systems 88

Chapter 8 VxWorks Platforms

8.1 Notes for VxWorks 7 Platforms 99
8.2 Known Defects 99
8.3 Increasing the Stack Size 100
8.4 Enabling Floating Point Coprocessor in Kernel Tasks 100
8.5 Downloadable Kernel Modules (DKM) for Kernel Mode on VxWorks Systems 100
8.6 Libraries for RTP Mode on VxWorks Systems 101
8.7 Requirement for Restarting Applications 101
8.8 Support for Modern C++ API 102
8.8.1 How to build VxWorks applications that use both Boost and the Connext DDS Modern C++ API 102
8.9 Multicast Support 103
8.10 Supported Transports 103
8.10.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID 104
8.10.2 How To Run Connext DDS Libraries in Kernels Built without Shared Memory 104
8.11 Unsupported Features 105
8.12 Monotonic Clock Support 105
8.13 Use of Real-Time Clock 105
8.14 Thread Configuration 105
8.15 Libraries Required for Using Distributed Logger 107
8.16 Libraries Required for Using Monitoring 108
8.17 Libraries Required for Using RTI Real-Time WAN Transport APIs 109
8.18 Libraries Required for Zero Copy Transfer Over Shared Memory 109
8.19 Increasing the Receive Socket Buffer Size 110

Chapter 9 Windows Platforms

9.1 Requirements when Using Microsoft Visual Studio 122
9.2 Linking with Libraries for Windows Platforms 123
9.3 Use Dynamic MFC Library, Not Static 124
9.4 .NET API Requires Thread Affinity 124

9.5 Support for Modern C++ API 124
9.6 Multicast Support 125
9.7 Supported Transports 125
9.8 Unsupported Features 125
9.9 Monotonic Clock Support 125
9.10 Thread Configuration 126
9.11 Support for 'Find Package' CMake Script 127
9.12 Durable Writer History and Durable Reader State Features 127
9.13 Backtrace Support 128
9.14 Support for Remote Procedure Calls (RPC) 128
9.15 Libraries Required for Using Distributed Logger Support 128
9.16 Libraries Required for Using Monitoring 129
9.17 Libraries Required for Using RTI Real-Time WAN Transport APIs 130
9.18 Libraries Required for Using RTI Secure WAN Transport APIs 130
9.18.1 Location for OpenSSL Libraries 131
9.19 Libraries Required for Using RTI TCP Transport APIs 131
9.20 Libraries Required for Zero Copy Transfer Over Shared Memory 132
9.21 Domain ID Support 133

Chapter 1 Supported Platforms

This document provides platform-specific instructions on how to compile, link, and run *RTI® Connext® DDS* applications.

For each platform, this document provides information on:

- Supported operating systems and compilers
- Required *Connext DDS* and system libraries
- Required compiler and linker flags, and environment variables for running the application (if any)
- Details on how the *Connext DDS* libraries were built
- Modern C++ API support
- Multicast support
- Supported transports
- Monotonic clock support
- Thread configuration
- Durable Writer History and Durable Reader State features support
- Support for 'Find Package' CMake script
- Backtrace support
- Remote Procedure Call support
- Required libraries for using additional features (such as Distributed Logger, Monitoring, Real-Time WAN Transport, Secure WAN Transport, TLS Support, Zero Copy Transfer Over Shared Memory)

Table 1.1 Supported Platforms

Operating System	Version and CPU
Android™ (target only)	Android 9.0 on Arm v7 and v8
INTEGRITY® (target only)	INTEGRITY 11.0.4 on p4080 INTEGRITY 11.4.4 on x64
Linux® on Arm® CPUs (target only)	NI™ Linux 3 on Arm v7 Ubuntu® 18.04 LTS on Arm v7 and Arm v8 Ubuntu 22.04 LTS on Arm v8
Linux on Intel® CPUs (host and target)	CentOS™ 7.0 on x64 Red Hat® Enterprise Linux 7.0, 7.3, 7.5, 7.6, 8.0 on x64 SUSE® Linux Enterprise Server 12 SP2 on x64 Ubuntu 18.04 LTS, 20.04 LTS, 22.04 LTS on x64 POSIX-compliant platforms, made available with *RTI Connext TSS:* CentOS 7.0 on x64 Red Hat Enterprise Linux 7, 7.3, 7.5, 7.6, 8 on x64 Ubuntu 18.04 LTS, 20.04 LTS on x64
Linux on Intel CPUs (target only - no Java support)	Ubuntu 16.04 LTS
macOS on Arm CPUs (target only)	macOS 11 on Arm v8 (*Requires Rosetta® 2 during installation, not required during runtime.)*
macOS® on Intel CPUs (host and target)	macOS 10.13 - 10.15, 11, 12 on x64
QNX® (target only)	QNX Neutrino® 6.4.1 on x86 QNX Neutrino 6.5 on x86 QNX Neutrino 6.5 SP1 on Arm v7 QNX Neutrino 7.0.4 on x64 and Arm v8 QNX Neutrino 7.1 on x64 and Arm v8
VxWorks® (target only)	VxWorks 7.0 SR0510 on x64 VxWorks 7.0 SR0630 on x64 VxWorks 21.11 on x64
Windows® (host and target)	Windows 10 [a] and 11 on x64, when using Visual Studio® 2017, 2019, or 2022 Windows Server 2012 R2, Server 2016 on x64, when using Visual Studio 2015

[a] Per Microsoft, this should be compatible with Windows 10 IoT Enterprise with Windows native app.

The following table lists additional target libraries available for *Connext DDS*, for which RTI offers custom support. If you are interested in using one of these platforms, please contact your local RTI sales representative or email **sales@rti.com**.

Table 1.2 Custom Supported Platforms

Operating System	Version and CPU
AIX® (host and target)	AIX 7.2 on POWER9™
Linux on Arm CPUs (target only)	TI® Linux 8.2.0.3 on Arm v8
	Yocto Project® 2.5 on Arm v8
Linux® on Intel CPUs (target only)	RedHawk™ Linux 8.2.1 on x64
QNX® (target only)	QNX Neutrino 6.5 on PPC e500v2
	QNX Neutrino 6.6 on Arm v7 and x86
	QNX Neutrino 7.0.4 on Arm v7
VxWorks	VxWorks 7.0 SR0660 on Arm v8

1.1 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

  This refers to the installation directory for *RTI® Connext® DDS*. The default installation paths are:
  - macOS® systems:
    /Applications/rti_connext_dds-6.1.2
  - Linux systems, non-*root* user:
    /home/<*your user name*>/rti_connext_dds-6.1.2
  - Linux systems, *root* user:
    /opt/rti_connext_dds-6.1.2
  - Windows® systems, user without Administrator privileges:
    <*your home directory*>\rti_connext_dds-6.1.2
  - Windows systems, user with Administrator privileges:
    C:\Program Files\rti_connext_dds-6.1.2

  You may also see **$NDDSHOME** or **%NDDSHOME%**, which refers to an environment variable set to the installation path.

  Wherever you see **<NDDSHOME>** used in a path, replace it with your installation path.

  Note for Windows Users: When using a command prompt to enter a command that includes the path **C:\Program Files** (or any directory name that has a space), enclose the path in quotation marks. For example:

  ```
  "C:\Program Files\rti_connext_dds-6.1.2\bin\rtiddsgen"
  ```

  Or if you have defined the **NDDSHOME** environment variable:

  ```
  "%NDDSHOME%\bin\rtiddsgen"
  ```

- ***<path to examples>***

  By default, examples are copied into your home directory the first time you run *RTI Launcher* or any script in **<NDDSHOME>/bin**. This document refers to the location of the copied examples as ***<path to examples>***.

  Wherever you see ***<path to examples>***, replace it with the appropriate path.

  Default path to the examples:
  - macOS systems: **/Users/<*your user name*>/rti_workspace/6.1.2/examples**
  - Linux systems: **/home/<*your user name*>/rti_workspace/6.1.2/examples**

- Windows systems: **<*your Windows documents folder*>\rti_workspace\6.1.2\examples**

  Where 'your Windows documents folder' depends on your version of Windows. For example, on Windows 10, the folder is **C:\Users\<*your user name*>\Documents**.

Note: You can specify a different location for **rti_workspace**. You can also specify that you do not want the examples copied to the workspace. For details, see *Controlling Location for RTI Workspace and Copying of Examples* in the *RTI Connext DDS Installation Guide*.

Chapter 2 AIX Platforms

Chapter 2 AIX Platforms lists the architectures supported on the IBM® AIX® operating system.

Table 1 Supported AIX Platforms

Operating System	CPU	Compiler	RTI Architecture Abbreviation
AIX 7.2 *Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.*	POWER9™	xlclang 16.1	64p9AIX7.2xlclang16.1

Table 2.1 Building Instructions for AIX Architectures lists the compiler flags and the libraries you will need to link into your application.

See also:

- 2.5 Libraries Required for Using Distributed Logger on page 12
- 2.6 Libraries Required for Using Monitoring on page 12
- 2.7 Libraries Required for Zero Copy Transfer Over Shared Memory on page 12

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 2.2 Running Instructions for AIX Architectures provides details on the environment variables that must be set at run time for an AIX architecture.

Table 2.3 Library-Creation Details for AIX Architectures provides details on how the libraries were built. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

Table 2.1 Building Instructions for AIX Architectures

API	Library Format	Required RTI Libraries[a]	Required System Libraries	Required Compiler Flags
Traditional C++	Static Release	libnddscppz.a or libnddscpp2z.a libnddscz.a libnddscorez.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	N/A	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Static Debug	libnddscppzd.a or libnddscpp2zd.a libnddsczd.a libnddscorezd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a	N/A	-g -O -q64 -qminimaltoc -DCPU-U=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Dynamic Release	libnddscpp.so or libnddscpp2.so libnddsc.so libnddscore.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so	-ldl -lnsl -lm -pthread -brtl	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmkshrobj -brtl -bbigtoc -qthreaded
	Dynamic Debug	libnddscppd.so or libnddscpp2d.so libnddscd.so libnddscored.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so	-ldl -lnsl -lm -pthread -brtl	-g -O -q64 -qminimaltoc -DCPU-U=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmkshrobj -brtl -bbigtoc -qthreaded

[a] *Connext DDS* C/C++ libraries are in ${NDDSHOME}/lib/<*architecture*>. NDDSHOME is where *Connext DDS* is installed.

Table 2.1 Building Instructions for AIX Architectures

API	Library Format	Required RTI Libraries[a]	Required System Libraries	Required Compiler Flags
C	Static Release	libnddscz.a libnddscorez.a librticonnextmsgcz.a	N/A	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Static Debug	libnddsczd.a libnddscorezd.a librticonnextmsgczd.a	N/A	-g -O -q64 -qminimaltoc -DCPU- U=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Dynamic Release	libnddsc.so libnddscore.so librticonnextmsgc.so	-ldl -lnsl -lm -pthread -brtl	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmkshrobj -brtl -bbigtoc -qthreaded
	Dynamic Debug	libnddscd.so libnddscored.so librticonnextmsgcd.so	-ldl -lnsl -lm -pthread -brtl	-g -O -q64 -qminimaltoc -DCPU- U=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmkshrobj -brtl -bbigtoc -qthreaded

Table 2.2 Running Instructions for AIX Architectures

RTI Architecture	Library Format (Release & Debug)	Required Environment Variables[b]
64p9AIX7.2xlclang16.1	Static	EXTSHM=ON
	Dynamic	LIBPATH=$(NDDSHOME)/lib/<*arch*>: $(LIBPATH) EXTSHM=ON

Table 2.3 Library-Creation Details for AIX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
64p9AIX7.2xlclang16.1	Release	-O -q64 -qminimaltoc -D_LINUX_SOURCE_COMPAT -D_EXTENSIONS_ -DCPU=Power9 -O -fPIC -qpic=large -qthreaded -qalias=noansi -G -qmkshrobj -brtl -DNDEBUG -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Debug	-g -O -q64 -qminimaltoc -D_LINUX_SOURCE_COMPAT -D_EXTENSIONS_ -DCPU=Power9 -O -fPIC -qpic=large -qthreaded -qalias=noansi -G -qmkshrobj -brtl -DRTI_AIX -DRTI_UNIX -DRTI_64BIT

[a] *Connext DDS* C/C++ libraries are in ${NDDSHOME}/lib/<*architecture*>. NDDSHOME is where *Connext DDS* is installed.

[b] See Notes for Using Shared Memory.

2.1 Multicast Support

Multicast is supported on all AIX platforms and is configured out of the box. That is, the default value for the initial peers list (NDDS_DISCOVERY_PEERS) includes a multicast address. See the API Reference HTML documentation for more information.

2.2 Transports

- **Shared memory:** Supported and enabled by default.
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Not supported.
- **TCP/IPv4:** Not supported.

2.2.1 Notes for Using Shared Memory

By default, the maximum number of shared memory segments you can use with AIX is quite small and limits the capability of *Connext DDS* applications to work properly over shared memory. To increase the maximum number of shared memory segments an application can use, set the following environment variable before invoking your *Connext DDS* application:

```
EXTSHM=ON
```

This environment variable is not required if your application does not use the shared memory transport.

To see a list of shared memory resources in use, please use the '**ipcs**' command. To clean up shared memory and shared semaphore resources, please use the '**ipcrm**' command.

The shared memory keys used by *Connext DDS* are in the range of 0x400000. For example:

```
ipcs -m | grep 0x004
```

The shared semaphore keys used by *Connext DDS* are in the range of 0x800000; the shared mutex keys are in the range of 0xb00000. For example:

```
ipcs -s | grep 0x008
ipcs -s | grep 0x00b
```

Please refer to the shared-memory transport online documentation for details on the shared memory and semaphore keys used by *Connext DDS*.

2.3 Unsupported Features

These features are not supported on AIX platforms:

- Backtrace
- Controlling CPU Core Affinity

- Durable Writer History and Durable Reader State
- 'Find Package' CMake script
- Internal setting of thread names at the operating-system level
- Modern C++ API
- Monotonic clock

2.4 Thread Configuration

See Table 2.4 Thread Settings for AIX Platforms and Table 2.5 Thread-Priority Definitions for AIX Platforms.

Table 2.4 Thread Settings for AIX Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	OS default thread priority
	stack_size	192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	OS default thread priority
	stack_size	192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_ FLOATING_POINT
	priority	OS default thread priority
	stack_size	4*192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 2.4 Thread Settings for AIX Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_ FLOATING_POINT
	priority	OS default thread priority
	stack_size	4*192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 2.5 Thread-Priority Definitions for AIX Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	If any of these constants are used to set the priority of the thread in the QoS, the OS's default thread priority will be used.
THREAD_PRIORITY_HIGH	
THREAD_PRIORITY_ABOVE_NORMAL	
THREAD_PRIORITY_NORMAL	
THREAD_PRIORITY_BELOW_NORMAL	
THREAD_PRIORITY_LOW	

2.4.1 Changing Thread Priority

Due to the AIX threading-model implementation, there are situations that require you to run your *Connext DDS* application with root privileges:

- **For all APIs:** Your application must have *root* privileges to use the thread option, DDS_ THREAD_SETTINGS_REALTIME_PRIORITY, for the event and receiver pool thread QoS (**DDS_DomainParticipantQos.event.thread, DDS_DomainParticipantQos.receiver_pool.-thread**).
- **For the Java API only:** Your application must have *root* privileges to change the event and receiver pool thread priorities **(DDS_DomainParticipantQos.event.thread, DDS_DomainParticipantQos.receiver_pool.thread**).

2.5 Libraries Required for Using Distributed Logger

To use the Distributed Logger APIs, link against the additional libraries in Table 2.6 Additional Libraries for using RTI Distributed Logger .

Table 2.6 Additional Libraries for using RTI Distributed Logger

Language	Static Release	Static Debug	Dynamic Release	Dynamic Debug
C	librtidlcz.a	librtidlczd.a	librtidlc.so	librtidcd.so
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a	librtidlc.so librtidlcpp.so	librtidlcd.so librtidlcppd.so

2.6 Libraries Required for Using Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the static release version of the *Connext DDS* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

If you are statically linking your application with DDS libraries and you want to add monitoring to your application, you will also need to statically link the monitoring library. The library cannot be loaded dynamically strictly through the QoS profile because it also depends on DDS to publish its data. Therefore, it depends on DDS; the DDS functionality would cause duplicate symbols to be found resulting, in the termination of the process.

Notes:

- Memory and CPU usage is not available in monitoring data.
- If you plan to use *static* libraries, the RTI library from Table 2.7 Additional Libraries for Using Monitoring must appear *first* in the list of libraries to be linked.

Table 2.7 Additional Libraries for Using Monitoring

Static Release	Static Debug	Dynamic Release	Dynamic Debug
librtimonitoringz.a	librtimonitoringzd.a	librtimonitoring.so	librtimonitoringd.so

2.7 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy transfer over shared memory feature, link against the additional library in Table 2.8 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 2.8 Additional Libraries for Zero Copy Transfer Over Shared Memory

Static Release	Static Debug	Dynamic Release	Dynamic Debug
libnddsmetpz.a	libnddsmetpzd.a	libnddsmetp.so	libnddsmetpd.so

Chapter 3 Android Platforms

Table 3.1 Supported Android Target Platforms lists the architectures supported on the Android operating system.

Table 3.1 Supported Android Target Platforms

Operating System	CPU	Compiler[a]	RTI Architecture Abbreviation
Android 9.0	Arm v7	Clang 8.0 (NDK r19b)	armv7Android9.0clang8.0ndkr19b
		Java® Platform, Standard Edition JDK 11	
	Arm v8	Clang 8.0 (NDK r19b)	arm64Android9.0clang8.0ndkr19b
		Java Platform, Standard Edition JDK 11	

See Table 3.2 Building Instructions for Android Architectures for a list of the compiler flags and libraries you will need to link into your application. For other libraries you may need, see:

- 3.8 Libraries Required for Using Distributed Logger on page 21
- 3.9 Libraries Required for Using Monitoring on page 22
- 3.10 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 22
- 3.11 Libraries Required for Using RTI Secure WAN Transport APIs on page 23
- 3.12 Libraries Required for Using RTI TCP Transport and TLS Support APIs on page 23

Make sure you are consistent in your use of debug and release versions of the libraries. Do not mix release and debug libraries.

[a]For Java: Dalvik VM is JDK 1.5 with some features from 1.6 (See Android documentation for details.)

Table 3.3 Running Instructions for Android Architectures provides details on the environment variables that must be set at run time for an Android architecture.

Table 3.4 Library-Creation Details for Android Architectures provides details on how the libraries were built. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

Connext DDS supports the Android operating system as a *target* platform. The target can be in one of two configurations: a consumer device (e.g., a Google™ Nexus™ 7 tablet) or as a "raw" Linux distribution. Building applications for the target occurs on a development machine using an Android SDK and, for C/C++, an Android NDK.

For a consumer device, all programs (applications and DDS utilities) must be installed on the device as Apps (***.apk** files). All Android Apps are loaded and executed by an instance of the Dalvik VM running as a Linux process. No *Connext DDS* components or libraries have to be pre-installed on the device—that is taken care of by the Android build and packaging tools. See the Android documentation for a full description of building and packaging Android Apps.

For a raw Linux distribution, all programs are executables that are linked with the necessary *Connext DDS* libraries (see Table 3.1 Supported Android Target Platforms). The build process is similar to other Linux variants, see Linux Platforms in the *Building Applications* chapter in the RTI Connext DDS Core Libraries User's Manual.

'Release' and 'Debug' Terminology:

Android and *Connext DDS* use these terms differently. For Android, "release" and "debug" refer to how application packages (***.apk**) are signed as part of the Android Security Model. A "release" package is cryptographically signed by a key that can be trusted by virtue of some certificate chain. A "debug" package is signed by a key distributed with the SDK. It says nothing about the origin of the package. It allows the package to be installed during development testing, hence "debug." For *Connext DDS*, debug means libraries created with debug symbols to facilitate debugging with gdb, for example. A "release" library does not contain debug information.

Additional Documentation:

See the RTI Connext DDS Core Libraries Getting Started Guide Addendum for Android Systems.

Table 3.2 Building Instructions for Android Architectures

<table>
<tr><th>API</th><th>Library Format</th><th>Required RTI Libraries and JAR Files[ab]</th><th>Required System Libraries</th><th>Required Compiler Flags</th></tr>
<tr><td rowspan="2">C++ (Traditional and Modern APIs)</td><td>Dynamic Release</td><td>libnddscore.so
libnddsc.so
libnddscpp.so
or
libnddscpp2.so
librticonnextmsgcpp.so
libc++_shared.so</td><td rowspan="2">For Android 9 on Arm v7a platforms:
-sysroot=$(ANDROID_NDK_PATH)/platforms/android28/arch-arm -LL$(ANDROID_NDK_PATH)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/arm-linux-androideabi/28 -llog -lc -lm -L$(ANDROID_NDK_PATH)/sources/cxx-stl/llvm-libc++/libs/armeabi-v7a -lc++_shared [c]

For Android 9.0 on Arm v8a platforms:
-sysroot=$(ANDROID_NDK_PATH)/platforms/android28/arch-arm -LL$(ANDROID_NDK_PATH)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/arm-linux-androideabi/28 -llog -lc -lm -L$(ANDROID_NDK_PATH)/sources/cxx-stl/llvm-libc++/libs/arm64-v8a- -lc++_shared [d]</td><td>For Android 9 on Arm v7a platforms:
-march=armv7-a
-DRTI_LINUX
-DRTI_UNIX
-DRTI_ANDROID
-DRTI_ANDROID9

For Android 9 on Arm v8a platforms:
-march=arm64v8-a
-DRTI_LINUX
-DRTI_UNIX
-DRTI_ANDROID
-DRTI_ANDROID9
-DRTI_64BIT</td></tr>
<tr><td>Dynamic Debug</td><td>libnddscored.so
libnddscd.so
libnddscppd.so
or
libnddscpp2d.so
librticonnextmsgcppd.so
libc++_shared.so</td></tr>
<tr><td rowspan="2">C</td><td>Dynamic Release</td><td>libnddscore.so
libnddsc.so
librticonnextmsgc.so</td><td rowspan="2">For Android 9 on Arm v7a platforms:
-sysroot=$(ANDROID_NDK_PATH)/platforms/android28/arch-arm -LL$(ANDROID_NDK_PATH)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/arm-linuxandroideabi/28 -llog -lc -lm -L$(ANDROID_NDK_PATH)/sources/cxx-stl/llvm-libc++/libs/armeabi-v7a

For Android 9 on Arm v8a platforms:
-sysroot=$(ANDROID_NDK_PATH)/platforms/android28/arch-arm64 -LL$(ANDROID_NDK_PATH)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/arm-linuxandroideabi/28 -llog -lc -lm -L$(ANDROID_NDK_PATH)/sources/cxx-stl/llvm-libc++/libs/arm64-v8a</td><td>For Android 9 on Arm v7a platforms:
-march=armv7-a
-DRTI_LINUX
-DRTI_UNIX
-DRTI_ANDROID
-DRTI_ANDROID9

For Android 9 on Arm v8a platforms:
-march=arm64v8-a
-DRTI_LINUX
-DRTI_UNIX
-DRTI_ANDROID
-DRTI_ANDROID9
-DRTI_64BIT</td></tr>
<tr><td>Dynamic Debug</td><td>libnddscored.so
libnddscd.so
librticonnextmsgcd.so</td></tr>
</table>

[a]Choose libnddscpp*.* for the Traditional C++ API or libnddscpp2*.* for the Modern C++ API.

[b]The RTI C/C++/Java libraries are in $(NDDSHOME)/lib/<architecture>.

[c]The library **libc++_shared.so** is always required when using C++ RTI Connext libraries and must be included in your application APK.

[d]The library **libc++_shared.so** is always required when using C++ RTI Connext libraries and must be included in your application APK.

Table 3.2 Building Instructions for Android Architectures

API	Library Format	Required RTI Libraries and JAR Files[ab]	Required System Libraries	Required Compiler Flags
Java	Release	When not building Apps (*.apk): nddsjava.jar rticonnextmsg.jar When building Apps (*.apk): nddsjava.jar libnddsjava.so libnddscore.so libnddsc.so rticonnextmsg.jar	N/A	None required
	Debug	When not building Apps (*.apk): nddsjavad.jar rticonnextmsgd.jar When building Apps (*.apk): nddsjavad.jar libnddsjavad.so libnddscored.so libnddscd.so rticonnextmsgd.jar		

Table 3.3 Running Instructions for Android Architectures

RTI Architecture	Library Format	Required Environment Variables
All supported Android architectures when not using Java	App (*.apk)	None
	Dynamic	LD_LIBRARY_PATH=$LD_LIBRARY_PATH: <path-to-ndds-libs>
All supported Android architectures when using Java	App (*.apk)	None
	Dex	LD_LIBRARY_PATH=$LD_LIBRARY_PATH: <path-to-ndds-libs> CLASSPATH=<path-to-dex>/classes.dex

[a]Choose libnddscpp*.* for the Traditional C++ API or libnddscpp2*.* for the Modern C++ API.

[b]The RTI C/C++/Java libraries are in $(NDDSHOME)/lib/<architecture>.

Table 3.4 Library-Creation Details for Android Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
arm64Android9.0clang8.0ndkr19b	Release	--target=aarch64-none-linux-android28 -DLINUX -DPtrIntType=long -DTARGET-T=\"arm64Android9.0clang8.0ndkr19b\" -O2 -Wall -Wno-unknown-pragmas -Wno-deprecated-declarations -Wno-macro-redefined -Wno-tautological-pointer-compare -Wno-logical-not-parentheses -Wno-constant-conversion -Wno-return-type-c-linkage -Wno-deprecated-register -Wno-tautological-constant-out-of-range-compare -Wno-enum-conversion -Wno-format-security -Wno-switch-bool -Wno-instantiation-after-specialization -Wno-exceptions -Wno-non-literal-null-conversion -Wstrict-prototypes -Wunused-parameter -funwind-tables -no-canonical-prefixes -fexceptions -O2 -DNDEBUG -fPIC
	Debug	--target=aarch64-none-linux-android28 -DLINUX -DPtrIntType=long -DTARGET-T=\"arm64Android9.0clang8.0ndkr19b\" -O0 -Wall -Wno-unknown-pragmas -Wno-deprecated-declarations -Wno-macro-redefined -Wno-tautological-pointer-compare -Wno-logical-not-parentheses -Wno-constant-conversion -Wno-return-type-c-linkage -Wno-deprecated-register -Wno-tautological-constant-out-of-range-compare -Wno-enum-conversion -Wno-format-security -Wno-switch-bool -Wno-instantiation-after-specialization -Wno-exceptions -Wno-non-literal-null-conversion -Wstrict-prototypes -Wunused-parameter -funwind-tables -no-canonical-prefixes -fexceptions -O0 -g -fPIC
armv7Android9.0clang8.0ndkr19b	Release	--target=armv7-none-linux-androideabi28 -DLINUX -DPtrIntType=long -DTARGET-T=\"armv7Android9.0clang8.0ndkr19b\" -O2 -Wall -Wno-unknown-pragmas -Wno-deprecated-declarations -Wno-macro-redefined -Wno-tautological-pointer-compare -Wno-logical-not-parentheses -Wno-constant-conversion -Wno-return-type-c-linkage -Wno-deprecated-register -Wno-tautological-constant-out-of-range-compare -Wno-enum-conversion -Wno-format-security -Wno-switch-bool -Wno-instantiation-after-specialization -Wno-exceptions -Wno-non-literal-null-conversion -Wstrict-prototypes -Wunused-parameter -march=armv7-a -mthumb -mfpu=vfpv3-d16 -mfloat-abi=softfp -funwind-tables -no-canonical-prefixes -fexceptions -O2 -DNDEBUG -fPIC
	Debug	--target=armv7-none-linux-androideabi28 -DLINUX -DPtrIntType=long -DTARGET-T=\"armv7Android9.0clang8.0ndkr19b\" -O0 -Wall -Wno-unknown-pragmas -Wno-deprecated-declarations -Wno-macro-redefined -Wno-tautological-pointer-compare -Wno-logical-not-parentheses -Wno-constant-conversion -Wno-return-type-c-linkage -Wno-deprecated-register -Wno-tautological-constant-out-of-range-compare -Wno-enum-conversion -Wno-format-security -Wno-switch-bool -Wno-instantiation-after-specialization -Wno-exceptions -Wno-non-literal-null-conversion -Wstrict-prototypes -Wunused-parameter -march=armv7-a -mthumb -mfpu=vfpv3-d16 -mfloat-abi=softfp -funwind-tables -no-canonical-prefixes -fexceptions -O0 -g -fPIC
All supported Android architectures for Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g

3.1 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs.

The Modern C++ API is available for all Android platforms and has been tested with both C++03 and C++11.

Notes:

- Support for C++03 is deprecated starting with release 6.1.0, which is the last release that supports non-C++11-compliant compilers. After release 6.1.0, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator* starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

3.2 Multicast Support

Multicast is available on supported Android platforms and is configured out of the box. That is, the default value for the initial peers list (NDDS_DISCOVERY_PEERS) includes a multicast address. See the API Reference HTML documentation for more information. Multicast has not been tested for this release and so, though available, is not officially supported. This should be addressed in a future release.

3.3 Transports

- **Shared memory:** Not supported for this release. For a consumer device, shared memory communication between Apps is often not desirable.
- **UDPv4**: Supported and enabled by default.
- **UDPv6**: Supported for Android 9 platforms only.
- **TCP/IPv4**: Supported.

3.4 Unsupported Features

These features are not supported on Android platforms:

- Backtrace
- Controlling CPU Core Affinity
- 'Find Package' CMake script
- Distributed Logger (supported on Android 9 only)
- Durable Writer History and Durable Reader State
- Setting of thread names by *Connext DDS* at the operating-system level

- Using DDS_WireProtocolQosPolicyAutoKind's RTPS_AUTO_ID_FROM_MAC to calculate the GUID prefix is not supported.
- Zero Copy Transfer Over Shared Memory

3.5 Monotonic Clock Support

The monotonic clock (described in *Clock Selection* in the RTI Connext DDS Core Libraries User's Manual) is supported on all Android platforms.

3.6 Thread Configuration

See Table 3.5 Thread Settings for Android Platforms and Table 3.6 Thread-Priority Definitions for Android Platforms.

Table 3.5 Thread Settings for Android Platforms

Applicable Threads	DDS_ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	OS default thread priority
	stack_size	OS default stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	OS default thread priority
	stack_size	OS default stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	

Table 3.5 Thread Settings for Android Platforms

Applicable Threads	DDS_ThreadSettings_t	Platform-Specific Setting
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	

Table 3.6 Thread-Priority Definitions for Android Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	If any of these constants are used to set the priority of the thread in the QoS, the OS's default thread priority will be used.
THREAD_PRIORITY_HIGH	
THREAD_PRIORITY_ABOVE_NORMAL	
THREAD_PRIORITY_NORMAL	
THREAD_PRIORITY_BELOW_NORMAL	
THREAD_PRIORITY_LOW	

3.7 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on Android architectures.

See *Remote Procedure Calls (RPC)* in the RTI Connext DDS Core Libraries User's Manual.

3.8 Libraries Required for Using Distributed Logger

RTI Distributed Logger is supported on all the platforms in Table 3.1 Supported Android Target Platforms.

To use the Distributed Logger APIs, link against the additional libraries in Table 3.7 Additional Libraries for using RTI Distributed Logger.

Select the files appropriate for your chosen library format. Make sure you are consistent in your use of debug and release versions of the libraries. Do not mix release and debug libraries.

Table 3.7 Additional Libraries for using RTI Distributed Logger

Language	Dynamic	
	Release	Debug
C	librtidlc.so	librtidcd.so
C++ (Traditional API)	librtidlc.so librtidlcpp.so	librtidlcd.so librtidlcppd.so
Java	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

3.9 Libraries Required for Using Monitoring

Make sure you are consistent in your use of debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the release version of the *Connext DDS* libraries, you will need to also use the release version of the monitoring library. Do not mix release and debug libraries.

Table 3.8 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Dynamic Release	librtimonitoring.so
Dynamic Debug	librtimonitoringd.so

3.10 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the RTI Real-Time WAN Transport Installation Guide for details.

Using *Real-Time WAN Transport* requires using one of the libraries in Table 3.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

Table 3.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[a]
Dynamic Release	libnddsrwt.so
Dynamic Debug	libnddsrwtd.so

3.11 Libraries Required for Using RTI Secure WAN Transport APIs

RTI Secure WAN Transport is only available on specific architectures. See the RTI Secure WAN Transport Release Notes and RTI Secure WAN Transport Installation Guide for details.

To use the *Secure WAN Transport* APIs, link against the additional libraries in Table 3.10 Additional Libraries for Using RTI Secure WAN Transport APIs. Select the files appropriate for your chosen library format.

Table 3.10 Additional Libraries for Using RTI Secure WAN Transport APIs

Library Format	Secure WAN Transport Libraries[b]	OpenSSL Libraries[c]
Dynamic Release	libnddstransportwan.so libnddstransporttls.so	librtisslsupport.so
Dynamic Debug	libnddstransportwand.so libnddstransporttlsd.so	

3.12 Libraries Required for Using RTI TCP Transport and TLS Support APIs

To use the TCP Transport APIs, link against the additional libraries in Table 3.11 Additional Libraries for Using RTI TCP Transport APIs. If you are using *RTI TLS Support*, also link against the libraries in Table 3.12 Additional Libraries for Using RTI TCP Transport APIs with TLS Enabled. Select the files appropriate for your chosen library format.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

[c]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

Table 3.11 Additional Libraries for Using RTI TCP Transport APIs

Library Format	TCP Transport Libraries[a]
Dynamic Release	libnddstransporttcp.so
Dynamic Debug	libnddstransporttcpd.so

Table 3.12 Additional Libraries for Using RTI TCP Transport APIs with TLS Enabled

Library Format	TCP Transport Libraries[b]	OpenSSL Libraries[c]
Dynamic Release	libnddstls.so	librtisslsupport.so
Dynamic Debug	libnddstlsd.so	

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

[c]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

Chapter 4 INTEGRITY Platforms

Table 4.1 Supported INTEGRITY Target Platforms lists the architectures supported on the INTEGRITY® operating system[a].

Table 4.1 Supported INTEGRITY Target Platforms

Operating System	CPU	Compiler	IP Stack	RTI Architecture Abbreviation
INTEGRITY 11.0.4	P4080	Multi 6.1.4	GHNet2 v2	p4080Inty11.devtree-fsl-e500m-c.comp2013.5.4
INTEGRITY 11.4.4	x64	Multi 7.1.6	GHNet2	pentiumInty11.pcx64

Table 4.2 Building Instructions for INTEGRITY Architectures lists the compiler flags and the libraries you will need to link into your application.

> Make sure you are consistent in your use of release and debug versions of the libraries. Do not mix release and debug libraries.

See also:

- 4.1 Required Patch for INTEGRITY 11.0.4 on the next page
- 4.7 Libraries Required for Using Distributed Logger on page 31
- 4.8 Libraries Required for Using Monitoring on page 32
- 4.9 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 32
- 4.10 Libraries Required for Zero Copy Transfer Over Shared Memory on page 33

Table 4.3 Running Instructions for INTEGRITY Architectures provides details on the environment variables that must be set at run time for an INTEGRITY architecture.

---

[a]For use with Windows hosts, as supported by Green Hills Software.

Table 4.4 Library-Creation Details for INTEGRITY Architectures provides details on how the libraries were built. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

4.1 Required Patch for INTEGRITY 11.0.4

For INTEGRITY 11.0.4 platforms, you must install this patch from Green Hills Software:

p4080Inty11.devtree-fsl-e500mc.comp2013.5.4: **patch_8154.iff**, **patch_8155.iff**, **patch_8246.iff**

For more information on the patch, please contact your Green Hills Software representative.

Table 4.2 Building Instructions for INTEGRITY Architectures

API	Library Format	Required RTI Libraries[abc]	Required System Libraries[d]	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnddscorez.a libnddscz.a libnddscppz.a or libnddscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	libsocket.a libnet.a libposix.a	All others: -DRTI_INTY --exceptions
	Static Debug	libnddscorezd.a libnddsczd.a libnddscppzd.a or libnddscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a (libnddscorezd.dba) (libnddsczd.dba) (libnddscppzd.dba or libnddscpp2zd.dba) (librticonnextmsgczd.dba)		
C	Static Release	libnddscorez.a libnddscz.a librticonnextmsgcz.a		
	Static Debug	libnddscorezd.a libnddsczd.a librticonnextmsgczd.a (libnddscorezd.dba) (libnddsczd.dba) (librticonnextmsgczd.dba)		

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The *.dba files contain the debugging information. You can link without these, as long as they are located in the same directory as the matching *d.a file (so that the MULTI® IDE can find the debug information).

[c]The RTI C/C++ libraries are in $(NDDSHOME)/lib/<architecture>.

[d]Transports (other than the default IP transport) such as StarFabric may require linking in additional libraries. For further details, see the API Reference HTML documentation or contact support@rti.com.

Table 4.3 Running Instructions for INTEGRITY Architectures

RTI Architecture	Required Environment Variables
All INTEGRITY architectures	None

Table 4.4 Library-Creation Details for INTEGRITY Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
p4080Inty11.devtree-fsl-e500mc.comp2013.5.4	Static Release	-DPtrIntType=long -DTARGET=\"p4080Inty11.devtree-fsl-e500mc.comp2013.5.4\" -O2 --unknown_pragma_silent --link_once_templates -fexceptions --diag_error 223 --prototype_warnings --exceptions -bsp=devtree-fsl-e500mc
	Static Debug	-DPtrIntType=long -DTARGET=\"p4080Inty11.devtree-fsl-e500mc.comp2013.5.4\" -O0 --unknown_pragma_silent --link_once_templates -fexceptions --diag_error 223 --prototype_warnings --exceptions -bsp=devtree-fsl-e500mc
pentiumInty11.pcx64	Static Release	-DPtrIntType=long -DTARGET=\"pentiumInty11.pcx64\" -O2 --unknown_pragma_silent --link_once_templates -fexceptions --diag_error 223 -prefixed_msgs -x86_64 -cpu=pentium4 -bsp=pcx64
	Static Debug	-DPtrIntType=long -DTARGET=\"pentiumInty11.pcx64\" -O0 --unknown_pragma_silent --link_once_templates -fexceptions --diag_error 223 -prefixed_msgs -x86_64 -cpu=pentium4 -bsp=pcx64
pentiumInty11.pcx86-smp	Static Release	-DPtrIntType=long -DTARGET=\"pentiumInty11.pcx86-smp\" -O2 --unknown_pragma_silent --link_once_templates -fexceptions --diag_error 223 -prefixed_msgs -bsp=pcx86-smp
	Static Debug	-DPtrIntType=long -DTARGET=\"pentiumInty11.pcx86-smp\" -O0 --unknown_pragma_silent --link_once_templates -fexceptions --diag_error 223 -prefixed_msgs -bsp=pcx86-smp

4.2 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs.

The Modern C++ API is available for all INTEGRITY platforms and has been tested with C++03. The INTEGRITY 11.4.4 architecture (pentiumInty11.pcx64) has also been tested with C++11.

Notes:

- Support for C++03 is deprecated starting with release 6.1.0, which is the last release that supports non-C++11-compliant compilers. After release 6.1.0, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator* starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

4.3 Multicast Support

Multicast is supported on all INTEGRITY platforms.

4.4 Supported Transports

- **Shared memory:** Supported, enabled by default. To clean up shared memory resources, reboot the kernel.
- **UDPv4:** Supported, enabled by default.
- **UDPv6:** Not supported.
- **TCP/IPv4:** Not supported.

4.5 Unsupported Features

These features are not supported on any INTEGRITY platform:

- Backtrace
- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script
- Monotonic clock
- Remote Procedure Calls (an experimental feature)

4.6 Thread Configuration

See these tables:

- Table 4.5 Thread Settings for INTEGRITY Platforms
- Table 4.6 Thread-Priority Definitions for INTEGRITY Platforms

Table 4.5 Thread Settings for INTEGRITY Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	16
	stack_size	64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	60
	stack_size	64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_ FLOATING_POINT
	priority	80
	stack_size	4*64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO
	priority	100
	stack_size	4*64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 4.6 Thread-Priority Definitions for INTEGRITY Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	16
THREAD_PRIORITY_HIGH	120
THREAD_PRIORITY_ABOVE_NORMAL	100
THREAD_PRIORITY_NORMAL	90
THREAD_PRIORITY_BELOW_NORMAL	80
THREAD_PRIORITY_LOW	60

4.6.1 Socket-Enabled and POSIX-Enabled Threads are Required

On INTEGRITY platforms, *Connext DDS* internally relies on the POSIX API for many of its system calls. As a result, any thread calling *Connext DDS* must be POSIX-enabled. By default, the 'Initial' thread of an address space is POSIX-enabled, provided the address space has been linked with **libposix.a**. Additional user threads that call *Connext DDS* must be spawned from the Initial thread using **pthread_create**. Only then is the created thread also POSIX-enabled. Note that tasks created at build time using the Integrate utility are *not* POSIX-enabled.

Furthermore, threads calling *Connext DDS* must be socket-enabled. This can be achieved by calling **InitLibSocket()** before making any *Connext DDS* calls and calling **ShutdownLibSocket** before the thread terminates. Note that an Initial thread is, by default, socket-enabled when the address space is linked with **libsocket.a**. Please refer to the *INTEGRITY Development Guide* for more information.

4.7 Libraries Required for Using Distributed Logger

RTI Distributed Logger is supported on INTEGRITY 11.0.4 on a P4080 CPU (p4080Inty11.devtree-fsl-e500mc.comp2013.5.4) and INTEGRITY 11.4.4 on an x64 CPU (pentiumInty11.pcx64). It is not supported on other INTEGRITY platforms.

Table 4.7 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

Table 4.7 Additional Libraries for using RTI Distributed Logger

Language	Static	
	Release	Debug[a]
C	librtidlcz.a	librtidlczd.a (librtidlczd.dba)
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a (librtidlczd.dba) (librtidlcppzd.dba)

4.8 Libraries Required for Using Monitoring

Make sure you are consistent in your use of debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the release version of the *Connext DDS* libraries, you will need to also use the release version of the monitoring library.

Notes:

- The RTI library in Table 4.8 Additional Libraries for Using Monitoring must appear *first* in the list of libraries to be linked.
- Automatic loading of the dynamic monitoring library through QoS is not supported.
- Memory and CPU usage is not available in monitoring data.

Table 4.8 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[b]
Static Release	librtimonitoringz.a
Static Debug	librtimonitoringzd.a

4.9 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the RTI Real-Time WAN Transport Installation Guide for details.

---

[a]The *.dba files contain the debugging information. You can link without these, as long as they are located in the same directory as the matching *d.a file (so that the MULTI® IDE can find the debug information).

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

Using *Real-Time WAN Transport* requires using one of the libraries in Table 4.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

Table 4.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[a]
Static Release	libnddsrwtz.a
Static Debug	libnddsrwtzd.a

4.10 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, link against the additional library in Table 4.10 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 4.10 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries[b]
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

4.11 Running over IP Backplane on a Dy4 Champ-AVII Board

Connext DDS can run on all four CPUs, provided the following hold true:

- *Connext DDS* applications on CPUs B, C and D only exchange data with applications on a different CPU or off-board.
- The IP backplane and associated routing has been properly configured. *Connext DDS* has been tested with the following libraries built into the INTEGRITY kernel: **debug**, **res**, **load**, **socket**, **itcpip**, **lbp**, **queue**, **ifbp**, **idb**, **bsl**.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

4.12 Out-of-the-box Transport Compatibility with Other Connext DDS Platforms

Due to some default kernel parameters on INTEGRITY platforms, the default value for **message_size_max** for the UDPv4 transport, and the default values for **message_size_max**, **received_message_count_max**, and **recv_buffer_size** for the shared-memory transport, are different than those for other platforms. This will cause out-of-the-box compatibility issues that may result in lack of communication. The mismatch in transport configuration between INTEGRITY and other platforms applies to *Connext DDS* 5.1.0 and higher. For more information, see the "Transport Compatibility" section in the *RTI Connext DDS Core Libraries Release Notes* for 5.3.1.

To address the compatibility issues, you can change the default transport settings of other platforms to match those of the INTEGRITY platform. Alternatively, you can update the INTEGRITY kernel parameters as described below so that the INTEGRITY platform will support larger transport settings.

The directive, GM_IP_FRAG_ENTRY_MAX_SIZE, limits the size of UDP packets that can be sent and received by INTEGRITY platforms. For details on this directive, please see Section 5.4.2 in the networking.pdf manual provided with the INTEGRITY kernel. The default value of GM_IP_FRAG_ENTRY_MAX_SIZE is 9216 bytes (not 16,000 bytes as is stated in the INTEGRITY documentation), which is why the default **message_size_max** for all transports supported for INTEGRITY is 9216 bytes.

If you want to send UDP messages larger than 9k, you must increase the value of GM_IP_FRAG_ENTRY_MAX_SIZE and rebuild the kernel. (You may also have to reconfigure other kernel parameters such as the socket, stack, and heap sizes to accommodate the larger value for GM_IP_FRAG_ENTRY_MAX_SIZE.) Failing to increase this value will cause failures when sending large UDP packets, and in some cases the **sendto()** call will fail silently.

4.12.1 Smaller Shared-Memory Receive-Resource Queue Size

INTEGRITY's shared-memory pluggable transport uses the shared-memory POSIX API. This API is part of the standard INTEGRITY distribution and is shipped as a library. This library uses a hard-coded value for the total amount of memory that can be shared with an address space. This limits the overall buffer space that can be used by the *DomainParticipants* within the same address space to communicate over shared memory with other *DomainParticipants*.

To allow more *DomainParticipants* to run within the same address space, we reduced the default size of the queue for each receive resource of the shared memory transport. The queue size is reduced to eight messages. This change only applies to INTEGRITY architectures and this default value can be overwritten through the shared memory transport QoS.

4.12.2 Using Shared Memory on INTEGRITY Systems

Connext DDS uses the single address-space POSIX library to implement the shared-memory transport on INTEGRITY operating systems.

To use shared-memory, you must configure your system to include the POSIX shared-memory library. The **posix_shm_manager** must be running in an "AddressSpace" solely dedicated to it. After building any *Connext DDS* application that uses shared memory, you must use the **intex** utility (provided with the INTEGRITY development environment) to pack the application with multiple address-spaces: one (or more) to contain the *Connext DDS* application(s), and another one to contain the **posix_shm_manager**.

Connext DDS will run on a target without the **posix_shm_manager**, but the POSIX functions will fail and return **ENOSYS**, and the participants will fail to communicate through shared memory.

To include the POSIX Shared-Memory Manager in its own Address Space:

The project files generated by *rtiddsgen* for MULTI will create the shared-memory manager for you. Please follow these steps:

1. Specify the path to your INTEGRITY distribution in the **_default.gpj** top-level project file by adding the following line (modify it according to the path to your INTEGRITY distribution):

   ```
   -os_dir=/local/applications/integrity/integrity-11.4.4
   ```
2. Build the project.
3. Before running your *Connext DDS* application on a target, download the **posix_shm_manager** file (generated by the build) onto the target.

The POSIX Shared Memory Manager will start automatically after the download and your applications will be able to use shared memory.

Notes:

- Only *one* **posix_shm_manager** is needed on a particular target. INTEGRITY offers the option of building this **posix_shm_manager** *inside* the kernel. Please refer to the INTEGRITY documentation.
- If you are already using shared memory through the POSIX library, there may be a possible conflict.
- INTEGRITY has two different types of POSIX libraries: a single-address space one (or 'light') and another one (complete POSIX implementation). *Connext DDS* uses the first one, but will work if you are using the complete POSIX implementation.

4.12.3 Shared Memory Limitations on INTEGRITY Systems

If several applications are running on the same INTEGRITY node and are using shared memory, once an application is stopped, it cannot be restarted. When the application is stopped (gracefully or ungracefully),

any new application on the same domain index within the same DDS domain will fail to start until the shared memory manager is also restarted.

Additionally, if the application is stopped ungracefully, the remaining applications will print several error messages such as the following until *Connext DDS* purges the stopped application from its database:

```
Resource Manager send error = 0x9
```

This error message is logged from INTEGRITY's POSIX shared memory manager, *not* from *Connext DDS*. The error message is benign and will not prevent the remaining applications from communicating with each other or with application on other nodes.

The workaround is to either restart the stopped application with a different participant index or shut down all the other applications and the shared memory manager, then restart everything.

4.13 Using rtiddsping and rtiddsspy on INTEGRITY Systems

While the RTI libraries for INTEGRITY can be used with any BSP, providing the processor falls under the same category (for example, the ppc7400... RTI libraries can be used on any target with a PPC74xx processor), *rtiddsping* and *rtiddsspy* are provided as executables, and therefore are BSP-dependent. You will not be able to run them successfully on your target if it is not compatible with the BSP listed in the architecture name (such as pcx86-smp). In this case, you will need to re-integrate the *rtiddsping* and *rtiddsspy* applications. Please refer to your hardware documentation for peripheral compatibility across BSPs.

4.14 Issues with INTEGRITY Systems

4.14.1 Delay When Writing to Unreachable Peers

On INTEGRITY systems, if a publishing application's initial peers list includes a nonexistent (or simply unreachable) host, calls to **write()** may block for approximately 1 second.

This long block is caused by the stack trying to resolve the invalid/unreachable host. Most IP stacks do not block the sending thread because of this reason, and you may include invalid/unreachable hosts in your initial-peers list. If you find that your stack does block the sending thread, please consult your IP stack vendor on how to change its behavior. [RTI Issue ID CORE-1637]

4.14.2 Linking with 'libivfs.a' without a File System

If you link your application with **libivfs.a** and are using a system that does not have a file system, you may notice the application blocks for 2 seconds at start-up.

4.14.3 Compiler Warnings Regarding Unrecognized #pragma Directives

Building *Connext DDS* projects for INTEGRITY causes the compiler to produce several warnings about #pragma directives not recognized in some *Connext DDS* header files. For example:

```
Building default.bld
"C:/ndds/ndds.4.4x/include/ndds/dds_c/dds_c_infrastructure.h", line 926:
warning: unrecognized #pragma
  #pragma warning(push)
          ^
"C:/ndds/ndds.4.4x/include/ndds/dds_c/dds_c_infrastructure.h", line 927:
warning: unrecognized #pragma
  #pragma warning(disable:4190)
          ^
"C:/ndds/ndds.4.4x/include/ndds/dds_c/dds_c_infrastructure.h", line 945:
warning: unrecognized #pragma
  #pragma warning(pop)
          ^
```

These warnings do not compromise the final application produced and can be safely ignored.

Chapter 5 Linux Platforms

First, see the basic instructions for compiling Linux Platforms in the *Building Applications* chapter in the RTI Connext DDS Core Libraries User's Manual. The following tables provide supplemental information.

Table 5.1 Linux Platforms on Intel CPUs below and Table 5.2 Linux Platforms on Arm CPUs on the next page list the supported Linux architectures. Each of these architectures is for a specific combination of OS, CPU, and compiler.

Table 5.1 Linux Platforms on Intel CPUs

Operating System	CPU	Compiler	RTI Architecture Abbreviation[a]
CentOS 7.0	x64	gcc 4.8.2	x64Linux3gcc4.8.2 x64Linux3gcc4.8.2FACE_GP
		Java Platform, Standard Edition JDK 11	x64Linux3gcc4.8.2
Red Hat Enterprise Linux 7.0, 7.3, 7.5, 7.6	x64	gcc 4.8.2	x64Linux3gcc4.8.2 x64Linux3gcc4.8.2FACE_GP
		Java Platform, Standard Edition JDK 11	x64Linux3gcc4.8.2
Red Hat Enterprise Linux 8.0	x64	gcc 7.3.0	x64Linux4gcc7.3.0 x64Linux4gcc7.3.0FACE_GP
		Java Platform, Standard Edition JDK 11	x64Linux4gcc7.3.0
RedHawk Linux 8.2.1 *Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.*	x64	gcc 4.4.5	x64RedHawk8.2gcc8.3.1

[a]Architectures ending with FACE_GP are available with *RTI Connext TSS*. See 5.5 Limitations of FACE Architectures on page 48.

Table 5.1 Linux Platforms on Intel CPUs

Operating System	CPU	Compiler	RTI Architecture Abbreviation[a]
SUSE Linux Enterprise Server 12 SP2	x64	gcc 4.3.4	x64Linux2.6gcc4.3.4
		Java Platform, Standard Edition JDK 11	
Ubuntu 18.04 LTS, 20.04 LTS, 22.04 LTS	x64	gcc 7.3.0	x64Linux4gcc7.3.0 x64Linux4gcc7.3.0FACE_GP
		Java Platform, Standard Edition JDK 11	x64Linux4gcc7.3.0
Ubuntu 16.04 LTS	x86	gcc 5.4.0	i86Linux3gcc5.4.0
		Not supported on Java	

Table 5.2 Linux Platforms on Arm CPUs

Operating System	CPU	Compiler	RTI Architecture Abbreviation
NI Linux 3	Arm v7	gcc 4.4.1	armv7AngstromLinux3.2gcc4.4.1.cortex-a9 [b]
TI Linux 8.2.0.3 *Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.*	Arm v8	gcc 9.2.1	armv8Linux-armgcc9.2.1
Ubuntu 18.04 LTS	Arm v7	gcc 7.5.0	armv7Linux4gcc7.5.0
		Java Platform, Standard Edition JDK 11	
	Arm v8	gcc 7.3.0	armv8Linux4gcc7.3.0
		Java Platform, Standard Edition JDK 11	
Ubuntu 22.04	Arm v8	gcc 7.3.0	armv8Linux4gcc7.3.0
		Java Platform, Standard Edition JDK 11	

[a]Architectures ending with FACE_GP are available with *RTI Connext TSS*. See 5.5 Limitations of FACE Architectures on page 48.

[b]These libraries require a hardware FPU in the processor and are compatible with systems that have soft-float libc.

Table 5.2 Linux Platforms on Arm CPUs

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Yocto Project 2.5 *Custom-supported target platform.* *Contact your RTI sales representative or sales@rti.com for more information.*	Arm v8	gcc 7.3.0	armv8Linux4gcc7.3.0

Table 5.3 Building Instructions for Linux Architectures on the next page lists the compiler flags and libraries you will need to link into your application.

See these sections for other libraries you may need:

- 5.12 Libraries Required for Using Distributed Logger on page 52
- 5.13 Libraries Required for Using Monitoring on page 53
- 5.14 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 54
- 5.15 Libraries Required for Using RTI Secure WAN Transport APIs on page 54
- 5.16 Libraries Required for Using RTI TCP Transport and TLS Support APIs on page 55
- 5.17 Libraries Required for Zero Copy Transfer Over Shared Memory on page 56

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 5.3 Building Instructions for Linux Architectures

API	Library Format	Required RTI Libraries or Jar Files[ab]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnddscorez.a libnddscz.a libnddscppz.a or libnddscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-ldl -lm -lpthread -lrt For Yocto Project, also add: -L/usr/lib/nptl	For 64-bit architectures (except TI Linux and Yocto Project): -DRTI_LINUX -DRTI_UNIX -m64 For TI Linux: -DRTI_LINUX -DRTI_UNIX --sysroot=SYSROOT For Yocto Project: -DRTI_LINUX -DRTI_UNIX -m32 --sysroot =SYSROOT For 32-bit architectures: -DRTI_LINUX -DRTI_UNIX -m32 For any Linux with GCC 6 or higher: -no-pie (see **Note** below table) For Ubuntu 18.04 LTS on Arm v7: -march=armv7 -mthumb -mfloat-abi=hard -mabi=aapcs-linux -fun-wind-tables For all architectures, if you want backtrace information, also add: **Compiler flag:** -fno-omit-frame-pointer **Linker flag:** -rdynamic **Arm architectures:** -funwind-tables (see 5.10 Backtrace Support on page 51)
	Static Debug	libnddscorezd.a libnddsczd.a libnddscppzd.a or libnddscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnddscore.so libnddsc.so libnddscpp.so or libnddscpp2.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so		
	Dynamic Debug	libnddscored.so libnddscd.so libnddscppd.so or libnddscpp2d.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so		

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]RTI C/C++/Java libraries are in <NDDSHOME>/lib/<architecture>. The jar files are in <NDDSHOME>/lib/java.

Table 5.3 Building Instructions for Linux Architectures

<table>
<tr><th>API</th><th>Library Format</th><th>Required RTI Libraries or Jar Files[ab]</th><th>Required System Libraries</th><th>Required Compiler Flags</th></tr>
<tr><td rowspan="4">C</td><td>Static Release</td><td>libnddscorez.a
libnddscz.a
librticonnextmsgcz.a</td><td rowspan="4">-ldl -lm -lpthread -lrt

For Yocto Project, also add:
-L/usr/lib/nptl</td><td rowspan="4">For 64-bit architectures (except TI Linux and Yocto Project):
-DRTI_LINUX -DRTI_UNIX -m64
For TI Linux:
-DRTI_LINUX -DRTI_UNIX --sysroot=SYSROOT
For Yocto Project:
-DRTI_LINUX -DRTI_UNIX -m32 --sysroot =SYSROOT

For 32-bit architectures:
-DRTI_LINUX -DRTI_UNIX -m32

For any Linux with GCC 6 or higher:
-no-pie (see Note below table)

For Ubuntu 18.04 LTS on Arm v7:
-march=armv7 -mthumb -mfloat-abi=hard -mabi=aapcs-linux -funwind-tables

For all architectures, if you want backtrace information, also add:
Compiler flag:
-fno-omit-frame-pointer
Linker flag:
-rdynamic
Arm architectures:
-funwind-tables
(see 5.10 Backtrace Support on page 51)</td></tr>
<tr><td>Static Debug</td><td>libnddscorezd.a
libnddsczd.a
librticonnextmsgczd.a</td></tr>
<tr><td>Dynamic Release</td><td>libnddscore.so
libnddsc.so
librticonnextmsgc.so</td></tr>
<tr><td>Dynamic Debug</td><td>libnddscored.so
libnddscd.so
librticonnextmsgcd.so</td></tr>
<tr><td rowspan="2">Java</td><td>Release</td><td>nddsjava.jar
rticonnextmsg.jar</td><td rowspan="2">N/A</td><td rowspan="2">None required</td></tr>
<tr><td>Debug</td><td>nddsjavad.jar
rticonnextmsgd.jar</td></tr>
</table>

Note: For any Linux with GCC 6 or higher: Starting with GCC 6, it's possible to configure the compiler driver to link, by default, executables with PIE (position independent executable) support on the amd64 and ppc64el architectures. Depending on the distributor of the GCC package, the automatic PIE generation may or may not be enabled. PIE executables cannot be used with RTI's libraries, due to Address Space Layout Randomization (ASLR). For this reason, RTI has linked Linux executables using

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]RTI C/C++/Java libraries are in <NDDSHOME>/lib/<architecture>. The jar files are in <NDDSHOME>/lib/java.

the **-no-pie** flag when the GCC version is 6 or higher. If you are using GCC 6 or newer, you must link the executable with **-no-pie** to prevent PIE generation.

Table 5.4 Running Instructions for Linux Architectures below provides details on the environment variables that must be set at run time for a Linux architecture.

Table 5.4 Running Instructions for Linux Architectures

RTI Architecture	Library Format	Environment Variables
All supported Linux architectures when using Java	N/A	LD_LIBRARY_PATH= ${NDDSHOME}/lib/<*architecture*>: ${LD_LIBRARY_PATH} **Note:** For all 64-bit Java architectures (...64Linux...), use **-d64** in the command line.
All supported Linux architectures when not using Java	Static (Release & Debug)	None required
	Dynamic (Release & Debug)	LD_LIBRARY_PATH= ${NDDSHOME}/lib/<*architecture*>: ${LD_LIBRARY_PATH}

Table 5.5 Library-Creation Details for Linux Architectures below provides details on how the Linux libraries were built. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

Table 5.5 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv7Angstrom Linux3.2 gcc4.4.1.cortex-a9	Static Release	-O -march=armv7-a -mtune=cortex-a9 -mfloat-abi=softfp -mfpu=vfpv3 -mthumb -mlong-calls -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv7AngstromLinux3.2gcc4.4.1.cortex-a9\"
	Static Debug	-O0 -march=armv7-a -mtune=cortex-a9 -mfloat-abi=softfp -mfpu=vfpv3 -mthumb -mlong-calls -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv7AngstromLinux3.2gcc4.4.1.cortex-a9\"
	Dynamic Release	-O -march=armv7-a -mtune=cortex-a9 -mfloat-abi=softfp -mfpu=vfpv3 -mthumb -mlong-calls -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv7AngstromLinux3.2gcc4.4.1.cortex-a9\"
	Dynamic Debug	-O0 -march=armv7-a -mtune=cortex-a9 -mfloat-abi=softfp -mfpu=vfpv3 -mthumb -mlong-calls -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv7AngstromLinux3.2gcc4.4.1.cortex-a9\"
armv7Linux4gcc7.5.0	Release (static and dynamic)	-Wall -Wno-unknown-pragmas -march=armv7 -mthumb -mfloat-abi=hard -mabi=aapcs-linux -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC
	Debug (static and dynamic)	-Wall -Wno-unknown-pragmas -march=armv7 -mthumb -mfloat-abi=hard -mabi=aapcs-linux -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC

Table 5.5 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv8Linux4gcc7.3.0	Static Release	-O -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv8Linux4gcc7.3.0\"
	Static Debug	-O0 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv8Linux4gcc7.3.0\"
	Dynamic Release	-O -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv8Linux4gcc7.3.0\"
	Dynamic Debug	-O0 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"armv8Linux4gcc7.3.0\"
armv8Linux-armgcc9.2.1	Static Release	-DLINUX -DPtrIntType=long -DTARGET="armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC
	Static Debug	-DLINUX -DPtrIntType=long -DTARGET="armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC
	Dynamic Release	-DLINUX -DPtrIntType=long -DTARGET="armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC
	Dynamic Debug	-DLINUX -DPtrIntType=long -DTARGET="armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC
x64Linux2.6gcc4.3.4	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux2.6gcc4.3.4\"
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux2.6gcc4.3.4\"
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux2.6gcc4.3.4\"
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux2.6gcc4.3.4\"
x64Linux3gcc4.8.2	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2\"
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2\"
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2\"
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2\"

Table 5.5 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Linux3gcc4.8.2FACE_GP	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux3gcc4.8.2FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
x64Linux4gcc7.3.0	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0\"
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0\"
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0\"
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0\"
x64Linux4gcc7.3.0FACE_GP	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET=\"x64Linux4gcc7.3.0FACE_GP\" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL

Table 5.5 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64RedHawk8.2gcc8.3.1	Static Release	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG
	Static Debug	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g
	Dynamic Release	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC
	Dynamic Debug	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC
i86Linux3gcc5.4.0	Static Release	-O -m32 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -DLINUX -DPtrIntType=long -DTARGET=\"i86Linux3gcc5.4.0\"
	Static Debug	-O0 -m32 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -DLINUX -DPtrIntType-e=long -DTARGET=\"i86Linux3gcc5.4.0\"
	Dynamic Release	-m32 -fPIC -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DPtrIntType-e=long -DCSREAL_IS_FLOAT -DCPU=i686 -DTARGET=\"i86Linux3gcc5.4.0\" -DNDEBUG
	Dynamic Debug	-m32 -fPIC -g -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i686 -DTARGET=\"i86Linux3gcc5.4.0\"
All supported Linux architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

5.1 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs. The Modern C++ API is available for all the platforms in Table 5.1 Linux Platforms on Intel CPUs on page 38 and Table 5.2 Linux Platforms on Arm CPUs on page 39 except the POSIX®-compliant architectures that end with "FACE_GP".

Some platforms have been tested only on C++03, whereas others have been tested with both C++03 and C++11. C++03 is typically supported with gcc3.4.2 and above. C++11 is typically supported with gcc4.7.2 and above.

Notes:

- Support for C++03 is deprecated starting with release 6.1.0, which is the last release that supports non-C++11-compliant compilers. After release 6.1.0, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator* starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

5.2 Multicast Support

Multicast is supported on all Linux platforms and is configured out of the box. That is, the default value for the initial peers list (**NDDS_DISCOVERY_PEERS**) includes a multicast address. See the API Reference HTML documentation for more information.

5.3 Supported Transports

- **Shared memory:** Supported and enabled by default. To clean up shared memory resources, reboot the kernel.
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported for all platforms except NI Linux 3.

  The UDPv6 transport is not enabled by default, and the peers list must be modified to support IPv6.

  Traffic Class support is only provided on architectures with gcc 4.1.0 or later that support the UDPv6 transport.
- **TCP/IPv4:** Supported for all Linux platforms except those for FACE TSS. This is *not* a built-in transport.

5.3.1 Shared Memory Support

To see a list of shared memory resources in use, please use the '**ipcs**' command. To clean up shared memory and shared semaphore resources, please use the '**ipcrm**' command.

The shared memory keys used by *Connext DDS* are in the range of 0x400000. For example:

```
ipcs -m | grep 0x004
```

The shared semaphore keys used by *Connext DDS* are in the range of 0x800000; the shared mutex keys are in the range of 0xb00000. For example:

```
ipcs -s | grep 0x008
ipcs -s | grep 0x00b
```

Please refer to the shared-memory transport online documentation for details on the shared memory and semaphore keys used by *Connext DDS*.

5.4 Unsupported Features

The setting of thread names at the operating-system level by *Connext DDS* is not supported on the following architectures:

- armv7AngstromLinux3.2gcc4.4.1.cortex-a9
- x64Linux2.6gcc4.3.4

Durable Writer History and Durable Reader State are not supported in this release.

5.5 Limitations of FACE Architectures

This section describes limitations when using the FACE architectures. These are POSIX-compliant architectures, which are made available with *RTI Connext TSS*:

- x64Linux3gcc4.8.2FACE_GP
- x64Linux4gcc7.3.0FACE_GP

The builtin shared memory transport of these architectures will not interoperate with non-FACE architectures.

When using the shared memory transport, shared memory resources may not be cleaned up by *Connext DDS*. Consequently, each application should clean up its own shared memory resources by removing the files in **/dev/shm/RTIOsapiSharedMemorySegment**.

The following features, utilities, and tools are not supported by the FACE architectures:

- Backtrace
- Cmake find package
- Distributed Logger
- Durable writer history and durable reader state
- Modern C++
- Monitoring
- Real-time clock
- Request/Reply communication pattern
- Remote Procedure Calls
- RTI DDS Ping, Spy, and Prototyper

- Setting thread names by *Connext DDS* at the operating-system level
- TCP transport

5.6 Monotonic Clock Support

The monotonic clock (described in *Clock Selection*, in the RTI Connext DDS Core Libraries User's Manual) is supported.

5.7 Thread Configuration

Table 5.6 Thread Settings for Linux Platforms below lists the thread settings for Linux platforms.

See also: Table 5.7 Thread-Priority Definitions for Linux Platforms on the next page and Table 5.8 Thread Kinds for Linux Platforms on the next page.

5.7.1 Support for Controlling CPU Core Affinity for RTI Threads

Support for controlling CPU core affinity (described in "Controlling CPU Core Affinity" in the User's Manual) is available on all supported Linux/SUSE platforms.

Table 5.6 Thread Settings for Linux Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	Empty CPU list (Supported on Linux and SUSE platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION (Supported on Linux and SUSE platforms)
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	Empty CPU list (Supported on Linux and SUSE platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION (Supported on Linux and SUSE platforms)

Table 5.6 Thread Settings for Linux Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_ POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	Empty CPU list (Supported on Linux and SUSE platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION (Supported on Linux and SUSE platforms)
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_ POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	Empty CPU list (Supported on Linux and SUSE platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION (Supported on Linux and SUSE platforms)

Table 5.7 Thread-Priority Definitions for Linux Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	If any of these constants are used to set the priority of the thread in the QoS, the OS's default thread priority will be used.
THREAD_PRIORITY_HIGH	
THREAD_PRIORITY_ABOVE_NORMAL	
THREAD_PRIORITY_NORMAL	
THREAD_PRIORITY_BELOW_NORMAL	
THREAD_PRIORITY_LOW	

Table 5.8 Thread Kinds for Linux Platforms

Thread Kinds	Operating-System Configuration[a]
DDS_THREAD_SETTINGS_FLOATING_POINT	N/A

[a]See the Linux programmer's manuals for more information.

Table 5.8 Thread Kinds for Linux Platforms

Thread Kinds	Operating-System Configuration[a]
DDS_THREAD_SETTINGS_STDIO	N/A
DDS_THREAD_SETTINGS_REALTIME_PRIORITY	Set schedule policy to SCHED_FIFO
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE	N/A

5.7.2 Using REALTIME_PRIORITY

If the **mask** field includes DDS_THREAD_SETTINGS_REALTIME_PRIORITY, a value must also be explicitly specified for the "priority" field in the QoS. (This is because using DDS_THREAD_SETTINGS_REALTIME_PRIORITY changes the scheduler used by Linux for the thread to SCHED_FIFO. If the **priority** field is not explicitly set, it will default to a value of 0, but this is an invalid value for a priority when using SCHED_FIFO.) Note that running with REALTIME_PRIORITY requires the appropriate privileges: the process will need to be run with root privileges on Linux in order to set the scheduler.

5.8 Durable Writer History and Durable Reader State Features

The Durable Writer History and Durable Reader State features are not supported in this release.

5.9 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on these Linux platforms:

- x64Linux2.6gcc4.3.4
- x64Linux4gcc7.3.0
- x64Linux3gcc4.8.2

For information on using this script, see *Building Applications Using CMake*, in the RTI Connext DDS Core Libraries Users Manual.

5.10 Backtrace Support

The Backtrace feature is supported on all Linux platforms except those for FACE TSS (x64Linux3gcc4.8.2FACE_GP and x64Linux4gcc7.3.0FACE_GP).

[a]See the Linux programmer's manuals for more information.

- If you are using GCC 6 or newer, recall you must link the executable with **-no-pie** to prevent PIE generation. See the **Note** below Table 5.3 Building Instructions for Linux Architectures.
- You will also need to compile with **-fno-omit-frame-pointer**.
- For Linux architectures on Arm CPUs, also use the **-funwind-tables** compiler option. This creates a table that allows the program to walk back through the function call stack from a given execution point.
- Symbol names may be unavailable without the use of special linker options. RTI has compiled Linux architectures using the linker option **-rdynamic** to display backtrace information. To display backtrace information on your Linux architecture, you must also compile with **-rdynamic**.
- 32-bit Windows architectures[a] are compiled using the additional **/Oy-** optimization flag.

See Logging a Backtrace for Failures, in the RTI Connext DDS Core Libraries User's Manual.

5.11 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on these Linux architectures:

- armv7Linux4gcc7.5.0
- armv8Linux-armgcc9.2.1
- armv8Linux4gcc7.3.0
- x64Linux4gcc7.3.0
- x64RedHawk8.2gcc8.3.1

See *Remote Procedure Calls (RPC)* in the RTI Connext DDS Core Libraries User's Manual.

5.12 Libraries Required for Using Distributed Logger

RTI Distributed Logger is supported on all the platforms in Table 5.1 Linux Platforms on Intel CPUs on page 38 and Table 5.2 Linux Platforms on Arm CPUs on page 39.

To use the Distributed Logger APIs, link against the additional libraries in Table 5.9 Additional Libraries for using RTI Distributed Logger .

[a]32-bit Windows architectures are available on demand, but are deprecated.

Table 5.9 Additional Libraries for using RTI Distributed Logger

Language	Static		Dynamic	
	Release	Debug	Release	Debug
C	librtidlcz.a	librtidlczd.a	librtidlc.so	librtidcd.so
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a	librtidlc.so librtidlcpp.so	librtidlcd.so librtidlcppd.so
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

5.13 Libraries Required for Using Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the static release version of the *Connext DDS* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

If you are statically linking your application with DDS libraries and you want to add monitoring to your application, you will also need to statically link the monitoring library. The library cannot be loaded dynamically strictly through the QoS profile because it also depends on DDS to publish its data. Therefore, it depends on DDS; the DDS functionality would cause duplicate symbols to be found resulting, in the termination of the process.

Note: If you plan to use *static* libraries, the RTI library in Table 5.10 Additional Libraries for Using Monitoring below must appear *first* in the list of libraries to be linked.

Table 5.10 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Dynamic Release	librtimonitoring.so
Dynamic Debug	librtimonitoringd.so
Static Release	librtimonitoringz.a
Static Debug	librtimonitoringzd.a

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

5.14 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the RTI Real-Time WAN Transport Installation Guide for details.

Using *Real-Time WAN Transport* requires one of the libraries in Table 5.11 Additional Libraries for Using RTI Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

Table 5.11 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[a]
Dynamic Release	libnddsrwt.so
Dynamic Debug	libnddsrwtd.so
Static Release	libnddsrwtz.a
Static Debug	libnddsrwtzd.a

5.15 Libraries Required for Using RTI Secure WAN Transport APIs

If you choose to use *RTI Secure WAN Transport*, it must be downloaded and installed separately.

It is only available for specific architectures. See the RTI Secure WAN Transport Release Notes and RTI Secure WAN Transport Installation Guide for details.(Or if it is not already installed, you can find the documentation here: https://community.rti.com/documentation).

To use the *Secure WAN Transport* APIs, link against the additional libraries in Table 5.12 Additional Libraries for using RTI Secure WAN Transport APIs on the next page. Select the files appropriate for your chosen library format.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

Table 5.12 Additional Libraries for using RTI Secure WAN Transport APIs

Library Format	RTI Secure WAN Transport Libraries[a]	OpenSSL Libraries[b]
Dynamic Release	libnddstransportwan.so libnddstransporttls.so	libssl.so libcrypto.so
Dynamic Debug	libnddstransportwand.so libnddstransporttlsd.so	
Static Release	libnddstransporttlsz.a libnddstransporttlszd.a	
Static Debug	libnddstransportwanz.a libnddstransportwanzd.a	

5.16 Libraries Required for Using RTI TCP Transport and TLS Support APIs

To use the TCP Transport APIs, link against the additional libraries in Table 5.13 Additional Libraries for using RTI TCP Transport APIs below.

If you are using *RTI TLS Support*, see Table 5.14 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled on the next page. Select the files appropriate for your chosen library format.

RTI TLS Support is an optional product for use with the TCP transport that is included with *RTI Connext® DDS*. If you choose to use *TLS Support*, it must be installed on top of a *Connext DDS* installation with the same version number; it can only be used on architectures that support TCP transport (see the RTI TLS Support Release Notes).

Table 5.13 Additional Libraries for using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries[c]
Dynamic Release	libnddstransporttcp.so
Dynamic Debug	libnddstransporttcpd.so
Static Release	libnddstransporttcpz.a
Static Debug	libnddstransporttcpzd.a

[a]The libraries are in <NDDSHOME>/lib/<architecture>.

[b]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

[c]These libraries are in <NDDSHOME>/lib/<architecture>.

Table 5.14 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[a]	OpenSSL Libraries[b]
Dynamic Release	libnddstls.so	libssl.so libcrypto.so
Dynamic Debug	libnddstlsd.so	
Static Release	libnddstlsz.a	
Static Debug	libnddstlszd.a	

5.17 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, link against the additional library in Table 5.15 Additional Libraries for Zero Copy Transfer Over Shared Memory below.

Table 5.15 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries[c]
Dynamic Release	libnddsmetp.so
Dynamic Debug	libnddsmetpd.so
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

[c]These libraries are in <NDDSHOME>/lib/<architecture>.

Chapter 6 macOS Platforms

Table 6.1 Supported macOS Platforms lists the architectures supported on macOS operating systems.

Table 6.1 Supported macOS Platforms

Operating System	CPU	Compiler	RTI Architecture Abbreviation
macOS 10.13, 10.14, 10.15, 11, 12	x64	clang 9.0, 10.0, 11.0, 12.0, 13	x64Darwin17clang9.0
		Java Platform, Standard Edition JDK 11	
macOS 11 Requires Rosetta®2 during installation. See 6.1 Installation Notes for macOS 11 Systems on Arm v8 CPUs on page 62	Arm v8	clang 12.0	arm64Darwin20clang12.0

Table 6.2 Building Instructions for macOS Architectures lists the compiler flags and libraries you will need to link into your application.

Other libraries you may need are described here:

- 6.12 Libraries Required for Using Distributed Logger on page 67
- 6.13 Libraries Required for Using Monitoring on page 68
- 6.14 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 68
- 6.15 Libraries Required for Using RTI Secure WAN Transport APIs on page 69
- 6.16 Libraries Required for Using RTI TCP Transport APIs on page 69
- 6.17 Libraries Required for Zero Copy Transfer Over Shared Memory on page 70

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 6.3 Running Instructions for macOS Architectures provides details on the environment variables that must be set at run time for a macOS architecture.

Table 6.4 Library-Creation Details for macOS Architectures provides details on how the libraries were built by RTI. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

Table 6.2 Building Instructions for macOS Architectures

API	Library Format	Required RTI Libraries[ab]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnddscorez.a libnddscz.a libnddscppz.a or libnddscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-ldl -lm -lpthread	For x64 architectures: -dynamic -lpthread -lc -single_module -DRTI_UNIX -DRTI_DARWIN -DRTI_64BIT For Arm v8 architectures: -DRTI_UNIX -DRTI_DARWIN
	Static Debug	libnddscorezd.a libnddsczd.a libnddscppzd.a or libnddscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnddscore.dylib libnddsc.dylib libnddscpp.dylib or libnddscpp2.dylib librticonnextmsgcpp.dylib or librticonnextmsgcpp2.dylib		
	Dynamic Debug	libnddscored.dylib libnddscd.dylib libnddscppd.dylib or libnddscpp2d.dylib librticonnextmsgcppd.dylib or librticonnextmsgcpp2d.dylib		

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The *Connext DDS* C/C++ libraries are in <NDDSHOME>/lib/<architecture>/.
<NDDSHOME> is where *Connext DDS* is installed, see 1.1 Paths Mentioned in Documentation on page 4

Table 6.2 Building Instructions for macOS Architectures

API	Library Format	Required RTI Libraries[ab]	Required System Libraries	Required Compiler Flags
C	Static Release	libnddscorez.a libnddscz.a librticonnextmsgcz.a	-ldl -lm -lpthread	-dynamic -lpthread -lc -single_module -DRTI_UNIX -DRTI_DARWIN -DRTI_64BIT
	Static Debug	libnddscorezd.a libnddsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnddscore.dylib libnddsc.dylib librticonnextmsgc.dylib		
	Dynamic Debug	libnddscored.dylib libnddscd.dylib librticonnextmsgcd.dylib		
Java	Release	nddsjava.jar rticonnextmsg.jar	N/A	None required
	Debug	nddsjavad.jar rticonnextmsgd.jar		

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The *Connext DDS* C/C++ libraries are in <NDDSHOME>/lib/<architecture>/.
<NDDSHOME> is where *Connext DDS* is installed, see 1.1 Paths Mentioned in Documentation on page 4

Table 6.3 Running Instructions for macOS Architectures

RTI Architecture	Library Format (Release & Debug)	Required Environment Variables[a]
arm64Darwin20clang12.0	Static	None required
	Dynamic	DYLD_LIBRARY_PATH=${NDDSHOME}/lib/arm64Darwin20clang12.0:${DYLD_LIBRARY_PATH}
x64Darwin17clang9.0	Static	None required
	Dynamic	DYLD_LIBRARY_PATH=${NDDSHOME}/lib/x64Darwin17clang9.0:${DYLD_LIBRARY_PATH}
x64Darwin17clang9.0 for Java	N/A	DYLD_LIBRARY_PATH=${NDDSHOME}/lib/x64Darwin17clang9.0:${DYLD_LIBRARY_PATH}

Table 6.4 Library-Creation Details for macOS Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
arm64Darwin20clang12.0	Release	-Dunix -O -Wall -Wno-unknown-pragmas -Wno-trigraphs -Wmissing-field-initializers -Wuninitialized -O -DNDEBUG -fPIC
	Debug	-Dunix -O0 -Wall -Wno-unknown-pragmas -Wno-trigraphs -Wmissing-field-initializers -Wuninitialized -O0 -g -fPIC
x64Darwin17clang9.0	Release	-arch x86_64 -mtune=core2 -Wno-trigraphs -fpascal-strings -fasm-blocks -O -Wall -Wno-unknown-pragmas -DPtrIntType=long -DTARGET=\"x64Darwin17clang9.0\" -DNDEBUG
	Debug	-arch x86_64 -mtune=core2 -Wno-trigraphs -fpascal-strings -fasm-blocks -g -O -Wall -Wno-unknown-pragmas -DPtrIntType=long -DTARGET=\"x64Darwin17clang9.0\"
x64Darwin17clang9.0 for Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g

[a]${NDDSHOME} is where *Connext DDS* is installed. ${DYLD_LIBRARY_PATH} represents the value of the DYLD_LIBRARY_PATH variable prior to changing it to support *Connext DDS*. When using nddsjava.jar, the Java virtual machine (JVM) will attempt to load release versions of the native libraries (nddsjava.dylib, nddscore.dylib, nddsc.dylib). When using nddsjavad.jar, the JVM will attempt to load debug versions of the native libraries (nddsjava.dylib, nddscore.dylib, nddsc.dylib).

6.1 Installation Notes for macOS 11 Systems on Arm v8 CPUs

Before installing the host and target bundles in macOS 11 (Big Sur) on Arm v8 (M1), Rosetta® 2 must be installed and enabled. Rosetta 2 is an Apple tool for translating third party software applications; without it, an error message will display when attempting to install *Connext DDS*. Installation instructions for Rosetta 2 can be found at https://support.apple.com/en-us/HT211861. Rosetta 2 is only required during *installation*. It is not required at runtime.

6.2 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs.

The Modern C++ API is available for all macOS platforms and has been tested with both C++03 and C++11.

Notes:

- Support for C++03 is deprecated starting with release 6.1.0, which is the last release that supports non-C++11-compliant compilers. After release 6.1.0, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator* starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

6.3 Multicast Support

Multicast is supported on macOS platforms and is configured out of the box. That is, the default value for the initial peers list (**NDDS_DISCOVERY_PEERS**) includes a multicast address. See the online documentation for more information.

6.4 Supported Transports

- **Shared memory:** Supported and enabled by default.
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported.
- **TCP/IPv4:** Supported.

6.5 Unsupported Features

These features are not supported on macOS platforms:

- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- Monotonic clock
- Find Package CMake script (on macOS 11 on Arm v8)

6.6 System Integrity Protection (SIP)

A feature called System Integrity Protection (SIP) was introduced in macOS 10.11. If enabled, this feature strips out the environment variable DYLD_LIBRARY_PATH, which is used to specify the location of shared libraries for a program. For more details, see https://support.apple.com/en-us/HT204899.

6.6.1 SIP and Java Applications

If you run *Connext DDS* applications using a Java Runtime Environment located under one of the paths protected by SIP (e.g., **/usr/bin**) and rely on the DYLD_LIBRARY_PATH environment variable to set the path to the *Connext DDS* run-time libraries (or any other third party run-time libraries, such as OpenSSL), Java will fail to load them with an error message such as:

```
The library libnddsjava.dylib could not be loaded by your operating system
```

To overcome this limitation, when running Java applications on macOS systems, you must use the **java.library.path** variable instead of the DYLD_LIBRARY_PATH environment variable to indicate the path to the *Connext DDS* libraries. This is automatically performed by the scripts to run applications generated by the *RTI Code Generator*. However, if you are manually running your *Connext DDS* application using the Java Runtime Environment, or you are writing our own scripts to run your Java application, you can indicate it as follows:

```
java -Djava.library.path="<installation_dir>/lib/<architecture>" -classpath .:"<installation_
dir>/lib/java/nddsjava.jar" <your_class>
```

Additionally, some *Connext DDS* applications may need to dynamically load functionality that is implemented in separate libraries (e.g., for the RTI Monitoring Library or transport plugins such as *RTI Secure WAN Transport* or *RTI TLS Support*). In that case, specifying the path to the **lib** directory using **java.library.path** is not sufficient, because the path to those libraries is not exposed to the underlying *Connext DDS* infrastructure.

To work around this limitation, you must provide the full path and extension of the dynamic libraries that are loaded at run time. In the case of the RTI Monitoring Library, this implies adding the following to your XML configuration file:

```
<domain_participant_qos>
    <property>
```

```
        <value>
            <element>
                <name>rti.monitor.library</name>
                <value>/full-path-to-librtimonitoring.dylib</value>
            </element>
            <!-- ... -->
        </value>
    </property>
</domain_participant_qos>
```

Likewise, for transport plugins that are loaded dynamically (e.g., the TCP transport plugin), you must add the full path to the XML configuration file:

```
<domain_participant_qos>
    <property>
        <!-- ... -->
        <value>
            <element>
                <name>dds.transport.TCPv4.tcp1.library</name>
                <value>/full-path-to-libnddstransporttcp.dylib</value>
            </element>
            <!-- ... -->
        </value>
    </property>
</domain_participant_qos>
```

For more on transport plugins, see 6.15 Libraries Required for Using RTI Secure WAN Transport APIs on page 69 and 6.16 Libraries Required for Using RTI TCP Transport APIs on page 69.

6.6.2 SIP and Connext Tools, Infrastructure Services, and Utilities

The SIP feature also makes it impossible for the scripts under **<installation_dir>/bin** to pick up the value of the DYLD_LIBRARY_PATH environment variable at run time. To workaround this issue, *Connext DDS* tools, infrastructure services, and utilities rely on RTI_LD_LIBRARY_PATH, an alternative environment variable that can be used in lieu of DYLD_LIBRARY_PATH and LD_LIBRARY_PATH to add library paths on Linux systems.

For example, to add **<OPENSSLHOME>/lib** and **<NDDSHOME/lib/<architecture>** (i.e., the library paths required for running *RTI Routing Service* with the Secure WAN or TLS transports) to your library path, you can export the RTI_LD_LIBRARY_PATH environment variable and run *Routing Service* as follows:

```
export RTI_LD_LIBRARY_PATH=<OPENSSLHOME>;/lib:<NDDSHOME>;/lib/<ARCHITECTURE>;
./<installation_dir>/bin/rtiroutingservice -cfgName <your_configuration>
```

6.7 Thread Configuration

See Table 6.5 Thread Settings for macOS Platforms and Table 6.6 Thread-Priority Definitions for macOS Platforms.

Table 6.5 Thread Settings for macOS Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_ FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_ FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 6.6 Thread-Priority Definitions for macOS Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	If any of these constants are used to set the priority of the thread in the QoS, the OS's default thread priority will be used.
THREAD_PRIORITY_HIGH	
THREAD_PRIORITY_ABOVE_NORMAL	
THREAD_PRIORITY_NORMAL	
THREAD_PRIORITY_BELOW_NORMAL	
THREAD_PRIORITY_LOW	

6.8 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on all macOS platforms in Table 6.1 Supported macOS Platforms except macOS 11 systems on Arm v8 CPUs.

For information on using this script, see *Building Applications Using CMake*, in the RTI Connext DDS Core Libraries Users Manual.

6.9 Backtrace Support

Backtrace is supported on macOS platforms and is configured out of the box. See Logging a Backtrace for Failures, in the RTI Connext DDS Core Libraries User's Manual.

6.10 Resolving NDDSUtility_sleep() Issues

When running on a macOS system, you may experience timing issues in your calls to **NDDSUtility_sleep()**. If you request to sleep for a small enough time period, you will notice that the actual sleep time is significantly longer.

macOS systems have a timer coalescing feature, enabled by default. This is a power-saving technique that reduces the precision of software timers, achieving a reduction in CPU usage.

What effect does this have on your *Connext DDS* application? Suppose you send samples from your publisher at a 5 ms rate, using **NDDSUtility_sleep()** to calculate that wait time. You have a subscriber with a deadline set to 6 ms. The timer coalescing feature could make your sleep last much longer than 5-6 ms, so when the next sample reaches the subscriber, the deadline period has expired and you will experience missed samples.

If you are having similar issues, see if your kernel has timer coalescing enabled. You can tell by using this command:

```
user@osx:~$ /usr/sbin/sysctl -a | grep coalescing_enabled
```

In the reply, a 1 means enabled, 0 means disabled.

```
kern.timer.coalescing_enabled: 1
```

To overcome this situation, you must disable timer coalescing in the kernel configuration. (Note that you must have **sudo** or **root** access to be able to edit this kernel parameter.)

```
user@osx:~$ sudo /usr/sbin/sysctl -w kern.timer.coalescing_enabled=0
```

The reply should be:

```
kern.timer.coalescing_enabled: 1 -> 0
```

This change won't be permanent though, and will go back to the default when the system is rebooted.

To make this change permanent, add the configuration line in the file **/etc/sysctl.conf**. You can use your favorite editor to do it, or use this command:

```
user@osx:~$ sudo echo "kern.timer.coalescing_enabled=0" >> /etc/sysctl.conf
```

6.11 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on all macOS platforms in Table 6.1 Supported macOS Platforms.

See *Remote Procedure Calls (RPC)* in the RTI Connext DDS Core Libraries User's Manual.

6.12 Libraries Required for Using Distributed Logger

RTI Distributed Logger is supported on macOS platforms. Table 6.7 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

Table 6.7 Additional Libraries for using RTI Distributed Logger

Language	Static		Dynamic	
	Release	Debug	Release	Debug
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a	librtidlc.dylib librtidlcpp.dylib	librtidlcd.dylib librtidlcppd.dylib
C	librtidlcz.a	librtidlczd.a	librtidlc.dylib	librtidlcd.dylib
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

6.13 Libraries Required for Using Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the static release version of the *Connext DDS* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

If you are statically linking your application with DDS libraries and you want to add monitoring to your application, you will also need to statically link the monitoring library. The library cannot be loaded dynamically strictly through the QoS profile because it also depends on DDS to publish its data. Therefore, it depends on DDS; the DDS functionality would cause duplicate symbols to be found resulting, in the termination of the process.

Note: If you are plan to use *static* libraries, the RTI library in Table 6.8 Additional Libraries for Using Monitoring must appear *first* in the list of libraries to be linked.

Table 6.8 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Dynamic Release	librtimonitoring.dylib
Dynamic Debug	librtimonitoringd.dylib
Static Release	librtimonitoringz.a
Static Debug	librtimonitoringzd.a

6.14 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the See the RTI Real-Time WAN Transport Installation Guide for details.

Using *Real-Time WAN Transport* requires one of the libraries in Table 6.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

Table 6.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[a]
Dynamic Release	libnddsrwt.dylib
Dynamic Debug	libnddsrwtd.dylib
Static Release	libnddsrwtz.a
Static Debug	libnddsrwtzd.a

6.15 Libraries Required for Using RTI Secure WAN Transport APIs

If you choose to use *RTI Secure WAN Transport*, it must be downloaded and installed separately. It is available on all macOS architectures. See the RTI Secure WAN Transport Release Notes and RTI Secure WAN Transport Installation Guide for details.

To use the Secure WAN Transport APIs, link against the additional libraries in Table 6.10 Additional Libraries for using RTI Secure WAN Transport APIs. Select the files appropriate for your chosen library format.

Table 6.10 Additional Libraries for using RTI Secure WAN Transport APIs

Library Format	RTI Secure WAN Transport Libraries[b]	OpenSSL Libraries[c]
Dynamic Release	libnddstransportwan.dylib libnddstransporttls.dylib	libssl.dylib libcrypto.dylib
Dynamic Debug	libnddstransportwand.dylib libnddstransporttlsd.dylib	
Static Release	libnddstransporttlsz.a libnddstransporttlszd.a	libsslz.a libcryptoz.a
Static Debug	libnddstransportwanz.a libnddstransportwanzd.a	

6.16 Libraries Required for Using RTI TCP Transport APIs

To use the TCP Transport APIs, link against the additional libraries in Table 6.11 Additional Libraries for using RTI TCP Transport APIs. If you are using *RTI TLS Support*, see Table 6.12 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled. Select the files appropriate for your chosen library format.

---

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

[c]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

Table 6.11 Additional Libraries for using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries[a]
Dynamic Release	libnddstransporttcp.dylib
Dynamic Debug	libnddstransporttcpd.dylib
Static Release	libnddstransporttcpz.a
Static Debug	libnddstransporttcpzd.a

Table 6.12 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[b]	OpenSSL Libraries[c]
Dynamic Release	libnddstls.dylib	libssl.dylib libcrypto.dylib
Dynamic Debug	libnddstlsd.dylib	
Static Release	libnddstlsz.a	libsslz.a libcryptoz.a
Static Debug	libnddstlszd.a	

6.17 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, link against the additional library in Table 6.13 Additional Libraries for Zero Copy Transfer Over Shared Memory .

Table 6.13 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Library
Dynamic Release	libnddsmetp.dylib
Dynamic Debug	libnddsmetpd.dylib
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

[c]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

Chapter 7 QNX Platforms

Table 7.1 Supported QNX Platforms lists the architectures supported on QNX operating systems.[a]

Table 7.1 Supported QNX Platforms

Operating System	CPU	Compiler	RTI Architecture
QNX Neutrino 6.4.1	x86	qcc 4.3.3 with GNU C++ libraries	i86QNX6.4.1qcc_gpp
QNX Neutrino 6.5	x86	qcc 4.4.2 with GNU C++ libraries	i86QNX6.5qcc_gpp4.4.2
	PowerPC™ E500 V2 core *Custom-supported target platform. Contact your RTI sales representative or sales@rti.-com for more information.*	qcc (gcc 4.4.2)	ppce500v2QNX6.5.0qcc_cpp4.4.2
QNX Neutrino 6.5 SP1	Arm v7	qcc 4.4.2 with Dinkumware libraries	armv7aQNX6.5.0SP1qcc_cpp4.4.2
QNX Neutrino 6.6.0	Arm v7 See note regarding patches below. *Custom-supported target platform. Contact your RTI sales representative or sales@rti.-com for more information.*	qcc_cpp 4.7.3 with Dinkumware libraries	armv7aQNX6.6.0qcc_cpp4.7.3
	x86 See note regarding patches below. *Custom-supported target platform. Contact your RTI sales representative or sales@rti.-com for more information.*	qcc_cpp 4.7.3 with Dinkumware libraries	i86QNX6.6qcc_cpp4.7.3

[a]For use with Windows or Linux hosts as supported by QNX and RTI.

Table 7.1 Supported QNX Platforms

Operating System	CPU	Compiler	RTI Architecture
QNX Neutrino 7.0.4	Arm v7 *Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.*	qcc 5.4.0 with LLVM default libraries	armv7QNX7.0.0qcc_cxx5.4.0 [a]
	Arm v8	qcc 5.4.0 with LLVM default libraries	armv8QNX7.0.0qcc_cxx5.4.0
		qcc 5.4.0 with GNU C++ libraries	armv8QNX7.0.0qcc_gpp5.4.0
	x64	qcc 5.4.0 with LLVM default libraries	x64QNX7.0.0qcc_cxx5.4.0
		qcc 5.4.0 with GNU C++ libraries	x64QNX7.0.0qcc_gpp5.4.0
QNX Neutrino 7.1	x64	qcc 8.3.0	x64QNX7.1qcc_cxx8.3.0
	Arm v8	qcc 8.3.0	armv8QNX7.1qcc_cxx8.3.0
		qcc 8.3.0	armv8QNX7.1qcc_gpp8.3.0 (see note below)
	Note: There are two different packages available for armv8QNX7.1qcc_gpp8.3.0: 1) A standard target platform that is compatible with OpenSSL 1.1.1n. 2) A custom-supported platform that is compatible with wolfSSL 4.7 when using RTI Security Plugins. Contact your RTI sales representative or sales@rti.com for more information.		

The libraries on Arm v7 and v8 CPUs require a hardware FPU in the processor and are compatible with systems that have hard-float libc. See Table 7.4 Library-Creation Details for QNX Architectures for compiler flags details.

Note regarding patches: The libraries for QNX Neutrino 6.6.0 on Arm v7 (armv7aQNX6.6.0qcc_cpp4.7.3), x86 (i86QNX6.6qcc_cpp4.7.3), and QNX Neutrino 7.0.4 on Arm v7 (armv7QNX7.0.0qcc_cxx5.4.0) were built with the following patches and tested on systems that have these patches installed:

- QNX Software Development Platform 6.6 Graphics Patch (Patch ID 3875)
- QNX Software Development Platform 6.6 Header Files Patch (Patch ID 3851)
- Patch-660-3885-diskimage.tar for the BSP

Table 7.2 Building Instructions for QNX Architectures lists the libraries you will need to link into your application.

---

[a] armv7QNX7.0.0qcc_cxx5.4.0 was tested with QNX Neutrino 7.0.0 kernel.

See also:

- 7.9 Libraries Required for Using Distributed Logger on page 84
- 7.10 Libraries Required for Using Monitoring on page 84
- 7.11 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 85
- 7.12 Libraries Required for Using RTI Secure WAN Transport APIs on page 86
- 7.13 Libraries Required for Using RTI TCP Transport APIs and TLS Support on page 86
- 7.14 Libraries Required for Zero Copy Transfer Over Shared Memory on page 87

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 7.3 Running Instructions for QNX Architectures provides details on the environment variables that must be set at run time for a QNX architecture.

Table 7.4 Library-Creation Details for QNX Architectures provides details on how the QNX libraries were built.

Starting with release 6.0.1, you will need the **dirname** tool to run the scripts in the **bin** directory.

Table 7.2 Building Instructions for QNX Architectures

API	Library Format	RTI Libraries[ab]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnddscorez.a libnddscz.a libnddscppz.a or libnddscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-lm -lsocket	-DRTI_QNX
	Static Debug	libnddscorezd.a libnddsczd.a libnddscppzd.a or libnddscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnddscore.so libnddsc.so libnddscpp.so or libnddscpp2.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so		
	Dynamic Debug	libnddscored.so libnddscd.so libnddscppd.so or libnddscpp2d.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so		

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The DDS C/C++ libraries are in $(NDDSHOME)/lib/<*architecture*>.

Table 7.2 Building Instructions for QNX Architectures

API	Library Format	RTI Libraries[ab]	Required System Libraries	Required Compiler Flags
C	Static Release	libnddscorez.a libnddscz.a librticonnextmsgcz.a	-lm -lsocket	-DRTI_QNX
	Static Debug	libnddscorezd.a libnddsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnddscore.so libnddsc.so librticonnextmsgc.so		
	Dynamic Debug	libnddscored.so libnddscd.so librticonnextmsgcd.so		

Table 7.3 Running Instructions for QNX Architectures

RTI Architecture	Library Format (Release & Debug)	Environment Variables
All supported QNX architectures	Static	None required
	Dynamic	LD_LIBRARY_PATH= ${NDDSHOME}/lib/<architecture>: ${LD_LIBRARY_PATH}[c]

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The DDS C/C++ libraries are in $(NDDSHOME)/lib/<*architecture*>.

[c]${NDDSHOME} represents the root directory of your *Connext DDS* installation. ${LD_LIBRARY_PATH} represents the value of the LD_LIBRARY_PATH variable prior to changing it to support *Connext DDS*. When using nddsjava.jar, the Java virtual machine (JVM) will attempt to load release versions of the native libraries. When using nddsjavad.jar, the JVM will attempt to load debug versions of the native libraries.

Table 7.4 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
armv7aQNX6.5.0SP1 qcc_cpp4.4.2	Static Release	-Vgcc/4.4.2,gcc_ntoarmv7le_cpp -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.5.0SP1qcc_cpp4.4.2\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -O -DNDEBUG
	Static Debug	-Vgcc/4.4.2,gcc_ntoarmv7le_cpp -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.5.0SP1qcc_cpp4.4.2\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -O0 -g
	Dynamic Release	-Vgcc/4.4.2,gcc_ntoarmv7le_cpp -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.5.0SP1qcc_cpp4.4.2\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/4.4.2,gcc_ntoarmv7le_cpp -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.5.0SP1qcc_cpp4.4.2\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -O0 -g -fPIC
armv7aQNX6.6.0 qcc_cpp4.7.3	Static Release	-Vgcc/4.7.3,gcc_ntoarmv7le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.6.0qcc_cpp4.7.3\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cpp -O -DNDEBUG
	Static Debug	-Vgcc/4.7.3,gcc_ntoarmv7le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.6.0qcc_cpp4.7.3\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cpp -O0 -g
	Dynamic Release	-Vgcc/4.7.3,gcc_ntoarmv7le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.6.0qcc_cpp4.7.3\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cpp -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/4.7.3,gcc_ntoarmv7le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv7aQNX6.6.0qcc_cpp4.7.3\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cpp -O0 -g -fPIC
armv7QNX7.0.0 qcc_cxx5.4.0	Static Release	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"armv7QNX7.0.0qcc_cxx5.4.0\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O -DNDEBUG
	Static Debug	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"armv7QNX7.0.0qcc_cxx5.4.0\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O0 -g
	Dynamic Release	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"armv7QNX7.0.0qcc_cxx5.4.0\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"armv7QNX7.0.0qcc_cxx5.4.0\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O0 -g -fPIC

Table 7.4 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
armv8QNX7.0.0 qcc_cxx5.4.0	Static Release	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_cxx5.4.0\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O -DNDEBUG
	Static Debug	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_cxx5.4.0\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O0 -g
	Dynamic Release	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_cxx5.4.0\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_cxx5.4.0\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O0 -g -fPIC
armv8QNX7.0.0 qcc_gpp5.4.0	Static Release	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O -DNDEBUG
	Static Debug	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O0 -g
	Dynamic Release	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/5.4.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET-T=\"armv8QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O0 -g -fPIC
armv8QNX7.1qcc_ cxx8.3.0	Static Release	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fPIC -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DNDEBUG -DRTI_QNX
	Static Debug	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fPIC -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DRTI_QNX
	Dynamic Release	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DNDEBUG -DRTI_QNX -fPIC
	Dynamic Debug	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DRTI_QNX -fPIC

Table 7.4 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
armv8QNX7.1 qcc_gpp8.3.0	Static Release	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fPIC -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DNDEBUG -DRTI_QNX
	Static Debug	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fPIC -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DRTI_QNX
	Dynamic Release	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DNDEBUG -DRTI_QNX -fPIC
	Dynamic Debug	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_cxx -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_cxx8.3.0\" -DRTI_QNX -fPIC
i86QNX6.4.1qcc_gpp	Static Release	-Vgcc/4.3.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.4.1qcc_gpp\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O -DNDEBUG
	Static Debug	-Vgcc/4.3.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.4.1qcc_gpp\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O0 -g
	Dynamic Release	-Vgcc/4.3.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.4.1qcc_gpp\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/4.3.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.4.1qcc_gpp\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O0 -g -fPIC
i86QNX6.5qcc_gp-p4.4.2	Static Release	-Vgcc/4.4.2,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.5qcc_gpp4.4.2\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -m32 -march=i386 -mtune=generic -O -DNDEBUG
	Static Debug	-Vgcc/4.4.2,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.5qcc_gpp4.4.2\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -m32 -march=i386 -mtune=generic -O0 -g
	Dynamic Release	-Vgcc/4.4.2,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.5qcc_gpp4.4.2\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -m32 -march=i386 -mtune=generic -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/4.4.2,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.5qcc_gpp4.4.2\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -m32 -march=i386 -mtune=generic -O0 -g -fPIC

Table 7.4 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
i86QNX6.6 qcc_cpp4.7.3	Static Release	-Vgcc/4.7.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.6qcc_cpp4.7.3\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cpp -m32 -march=i386 -mtune=generic -O -DNDEBUG
	Static Debug	-Vgcc/4.7.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.6qcc_cpp4.7.3\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cpp -m32 -march=i386 -mtune=generic -O0 -g
	Dynamic Release	-Vgcc/4.7.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.6qcc_cpp4.7.3\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cpp -m32 -march=i386 -mtune=generic -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/4.7.3,gcc_ntox86 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"i86QNX6.6qcc_cpp4.7.3\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cpp -m32 -march=i386 -mtune=generic -O0 -g -fPIC
ppce500v2QNX6.5.0 qcc_cpp4.4.2	Static Release	-Vgcc/4.4.2,gcc_ntoppcbespe -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"ppce500v2QNX6.5.0qcc_cpp4.4.2\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cpp -mcpu=8540 -me500v2 -mno-isel -mspe -mhard-float -Wl,--relax -O -DNDEBUG
	Static Debug	-Vgcc/4.4.2,gcc_ntoppcbespe -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"ppce500v2QNX6.5.0qcc_cpp4.4.2\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cpp -mcpu=8540 -me500v2 -mno-isel -mspe -mhard-float -Wl,--relax -O0 -g
	Dynamic Release	-Vgcc/4.4.2,gcc_ntoppcbespe -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"ppce500v2QNX6.5.0qcc_cpp4.4.2\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cpp -mcpu=8540 -me500v2 -mno-isel -mspe -mhard-float -Wl,--relax -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/4.4.2,gcc_ntoppcbespe -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"ppce500v2QNX6.5.0qcc_cpp4.4.2\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cpp -mcpu=8540 -me500v2 -mno-isel -mspe -mhard-float -Wl,--relax -O0 -g -fPIC
x64QNX7.0.0 qcc_cxx5.4.0	Static Release	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_cxx5.4.0\" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O -DNDEBUG
	Static Debug	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_cxx5.4.0\" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O0 -g
	Dynamic Release	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_cxx5.4.0\" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_cxx5.4.0\" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O0 -g -fPIC

Table 7.4 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
x64QNX7.0.0 qcc_gpp5.4.0	Static Release	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O -DNDEBUG
	Static Debug	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O0 -g
	Dynamic Release	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O -DNDEBUG -fPIC
	Dynamic Debug	-Vgcc/5.4.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET=\"x64QNX7.0.0qcc_gpp5.4.0\" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O0 -g -fPIC
x64QNX7.1qcc_ cxx8.3.0	Static Release	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DNDEBUG -DRTI_QNX -fPIC
	Static Debug	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64QNX7.1qcc_cxx8.3.0\" -DRTI_QNX -fPIC
	Dynamic Release	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DNDEBUG -DRTI_QNX -fPIC
	Dynamic Debug	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64QNX7.1qcc_cxx8.3.0\" -DRTI_QNX -fPIC

7.1 Required Change for Building with C++ Libraries for QNX Platforms

The C++ libraries are built *without* the **-fno-rtti** flag and *with* the **-fexceptions** flag. You must build your C++ applications *without* **-fno-exceptions** in order to link with the RTI libraries. In summary:

- Do *not* use **-fno-exceptions** when building a C++ application or the build will fail. It is not necessary to use **-fexceptions**, but doing so will not cause a problem.
- It is no longer necessary to use **-fno-rtti**, but doing so will not cause a problem.

7.2 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs.

Some platforms have been tested only on C++03, whereas others have been tested with both C++03 and C++11. C++03 is typically supported with gcc3.4.2 and above. C++11 is typically supported with gcc4.7.2 and above.

Notes:

- Support for C++03 is deprecated starting with release 6.1, which is the last release that supports non-C++11-compliant compilers. After release 6.1, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator* starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

7.3 Multicast Support

Multicast is supported on QNX platforms and is configured out of the box. That is, the default value for the initial peers list (NDDS_DISCOVERY_PEERS) includes a multicast address. See the online documentation for more information.

7.4 Supported Transports

- **Shared Memory:** Supported and enabled by default.

  To see a list of the shared memory resources, enter:

  ```
  'ls /dev/shmem/RTIOsapiSharedMemorySegment-*'
  ```

  To clean up the shared memory resources, remove the files listed in **dev/shmem/**. The shared resource names used by *Connext DDS* begin with **'RTIOsapiSharedMemorySem-**'. To see a list of shared semaphores, enter:

  ```
  'ls /dev/sem/RTIOsapiSharedMemorySemMutex*'
  ```

  To clean up the shared semaphore resources, remove the files listed in **/dev/sem/**'.

  The permissions for the semaphores created by *Connext DDS* are modified by the process' **umask** value. If you want to have shared memory support between different users, run the command "**umask 000**" to change the default **umask** value to 0 before running your *Connext DDS* application.

- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported. The transport is not enabled by default; the peers list must be modified to support IPv6. No Traffic Class support.

  To use the UDPv6 transport, the network stack must provide IPv6 capability. Enabling UDPv6 may involve switching the network stack server and setting up IPv6 route entries.
- **TCP/IPv4:** Supported on all platforms except:
  - i86QNX6.4.1qcc_gpp
  - ppce500v2QNX6.5.0qcc_cpp4.4.2

7.5 Unsupported Features

These features are not supported on QNX platforms:

- Backtrace
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script

7.6 Monotonic Clock Support

The monotonic clock (described in *Clock Selection* in the RTI Connext DDS Core Libraries User's Manual) is supported on all QNX platforms.

7.7 Thread Configuration

See Table 7.5 Thread Settings for QNX Platforms and Table 7.6 Thread-Priority Definitions for QNX Platforms.

Table 7.5 Thread Settings for QNX Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	10
	stack_size	64 * 1024
	cpu_list	Empty CPU list (Supported on QNX platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION

Table 7.5 Thread Settings for QNX Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	8
	stack_size	64 * 1024
	cpu_list	Empty CPU list (Supported on QNX platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	9
	stack_size	4 * 64 * 1024
	cpu_list	Empty CPU list (Supported on QNX platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	12
	stack_size	4 * 64 * 1024
	cpu_list	Empty CPU list (Supported on QNX platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION

Table 7.6 Thread-Priority Definitions for QNX Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	10
THREAD_PRIORITY_HIGH	14
THREAD_PRIORITY_ABOVE_NORMAL	12
THREAD_PRIORITY_NORMAL	10
THREAD_PRIORITY_BELOW_NORMAL	8
THREAD_PRIORITY_LOW	6

7.7.1 Support for Controlling CPU Core Affinity for RTI Threads

Support for controlling CPU core affinity (described in *Controlling CPU Core Affinity* in the RTI Connext DDS Core Libraries User's Manual) is available on all supported QNX platforms.

7.8 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on QNX 7.0.4 and higher architectures.

See *Remote Procedure Calls (RPC)* in the RTI Connext DDS Core Libraries User's Manual.

7.9 Libraries Required for Using Distributed Logger

RTI Distributed Logger is supported on all QNX platforms except QNX Neutrino 6.5 SP1 on Arm v7 (armv7aQNX6.5.0SP1qcc_cpp4.4.2).

Table 7.7 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

Table 7.7 Additional Libraries for using RTI Distributed Logger

Language	Static		Dynamic	
	Release	Debug	Release	Debug
C	librtidlcz.a	librtidlczd.a	librtidlc.so	librtidcd.so
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a	librtidlc.so librtidlcpp.so	librtidlcd.so librtidlcppd.so

7.10 Libraries Required for Using Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the static release version of the *Connext DDS* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

If you are statically linking your application with DDS libraries and you want to add monitoring to your application, you will also need to statically link the monitoring library. The library cannot be loaded dynamically strictly through the QoS profile because it also depends on DDS to publish its data. Therefore, it depends on DDS; the DDS functionality would cause duplicate symbols to be found resulting, in the termination of the process.

Notes:

- To use *static* libraries: the RTI library from Table 7.8 Additional Libraries for Using Monitoring must appear *first* in the list of libraries to be linked.
- To use *dynamic* libraries: make sure the permissions on the .so library files are readable by everyone.

Table 7.8 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Dynamic Release	librtimonitoring.so
Dynamic Debug	librtimonitoringd.so
Static Release	librtimonitoringz.a
Static Debug	librtimonitoringzd.a

7.11 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the RTI Real-Time WAN Transport Installation Guide for details.

Using *Real-Time WAN Transport* requires one of the libraries in Table 7.9 Additional Libraries for Using RTI Real-Time WAN Transport APISelect the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

Table 7.9 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[b]
Dynamic Release	libnddsrwt.so
Dynamic Debug	libnddsrwtd.so
Static Release	libnddsrwtz.a
Static Debug	libnddsrwtzd.a

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

7.12 Libraries Required for Using RTI Secure WAN Transport APIs

RTI Secure WAN Transport is only available specific QNX architectures, which are listed in the RTI Secure WAN Transport Release Notes.

If you choose to use *RTI Secure WAN Transport*, it must be downloaded and installed separately. See the RTI Secure WAN Transport Installation Guide.

To use the Secure WAN Transport APIs, link against the additional libraries in Table 7.10 Additional Libraries for using RTI Secure WAN Transport APIs on QNX Systems. (Select the files appropriate for your chosen library format.)

Table 7.10 Additional Libraries for using RTI Secure WAN Transport APIs on QNX Systems

Library Format	RTI Secure WAN Transport Libraries[a]	OpenSSL Libraries[b]
Dynamic Release	libnddstransportwan.so libnddstransporttls.so	libssl.so libcrypto.so
Dynamic Debug	libnddstransportwand.so libnddstransporttlsd.so	
Static Release	libnddstransporttlsz.a libnddstransporttlszd.a	
Static Debug	libnddstransportwanz.a libnddstransportwanzd.a	

7.13 Libraries Required for Using RTI TCP Transport APIs and TLS Support

To use the TCP Transport APIs, link against the additional libraries in Table 7.11 Additional Libraries for using RTI TCP Transport APIs .

Note: Not all platforms support the TCP Transport - see 7.4 Supported Transports on page 81.

Table 7.11 Additional Libraries for using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries[c]
Dynamic Release	libnddstransporttcp.so
Dynamic Debug	libnddstransporttcpd.so

[a]The libraries are in <NDDSHOME>/lib/<architecture>.

[b]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

[c]These libraries are in <NDDSHOME>/lib/<architecture>.

Table 7.11 Additional Libraries for using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries[a]
Static Release	libnddstransporttcpz.a
Static Debug	libnddstransporttcpzd.a

If you are using *RTI TLS Support*, also see Table 7.12 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled. (Select the files appropriate for your chosen library format.) See the RTI TLS Support Release Notes for a list of supported platforms.

Table 7.12 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[b]	OpenSSL Libraries[c]
Dynamic Release	libnddstls.so	libssl.so libcrypto.so
Dynamic Debug	libnddstlsd.so	
Static Release	libnddstlsz.a	
Static Debug	libnddstlszd.a	

7.14 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, link against the additional library in Table 7.13 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 7.13 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries[d]
Dynamic Release	libnddsmetp.so
Dynamic Debug	libnddsmetpd.so
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

[c]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-1.1.1n/<architecture>/<format>/lib.

[d]These libraries are in <NDDSHOME>/lib/<architecture>.

7.15 Restarting Applications on QNX Systems

Due to a limitation in the POSIX API, if a process is unexpectedly interrupted in the middle of a critical section of code that is protected by a shared mutex semaphore, the OS is unable to automatically release the semaphore, making it impossible to reuse it by another application.

The *Connext DDS* shared-memory transport uses a shared mutex to protect access to the shared memory area across multiple processes.

It is possible under some extreme circumstances that if one application crashes or terminates ungracefully while executing code inside a critical section, the other applications sharing the same resource will not be able to continue their execution. If this situation occurs, you must manually delete the shared-memory mutex before re-launching any application in the same DDS domain.

Chapter 8 VxWorks Platforms

Table 8.1 VxWorks Target Platforms lists the architectures supported on VxWorks operating systems. You can build a VxWorks application by cross-compiling from your development host.

Table 8.1 VxWorks Target Platforms

Operating System	CPU	Compiler	RTI Architecture[a]
VxWorks 7.0 SR0510	x64	gcc 4.8.1	For Kernel Modules: pentium64Vx7.0gcc4.8.1 For Real Time Processes: pentium64Vx7.0gcc4.8.1_rtp
VxWorks 7.0 SR0630	x64	llvm 8.0.0.2	For Kernel Modules: x64Vx7SR0630llvm8.0.0.2 For Real Time Processes: x64Vx7SR0630llvm8.0.0.2_rtp
VxWorks 7.0 SR0660 *Custom-supported target platform. Contact your RTI sales representative or sales@rti.com for more information.*	Arm v8	llvm 10.0.1.1	For Kernel Modules: armv8Vx7SR0660llvm10.0.1.cortex-a53 For Real Time Processes: armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp
VxWorks 21.11	x64	llvm 12.0.1.1	For Kernel Modules: x64Vx21.11llvm12.0.1.1 For Real Time Processes: x64Vx21.11llvm12.0.1.1_rtp

Table 8.2 Building Instructions for VxWorks Architectures on page 91 lists the libraries you will need to link into your application and the required compiler flags.

[a]For use with Windows hosts as supported by Wind River Systems.

For other libraries that you may need, see:

- 8.15 Libraries Required for Using Distributed Logger on page 107
- 8.16 Libraries Required for Using Monitoring on page 108
- 8.17 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 109
- 8.18 Libraries Required for Zero Copy Transfer Over Shared Memory on page 109

> Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 8.3 Running Instructions for VxWorks Architectures on page 92 provides details on the environment variables that must be set at run time for a VxWorks architecture.

Compiling a *Connext DDS* application for VxWorks depends on the development platform. For more information, such as specific compiler flags, see the *VxWorks Programmer's Guide*. Table 8.4 Library-Creation Details for VxWorks Architectures on page 93 provides details on how the VxWorks libraries were built. We recommend that you use similar settings.

Cross-compiling for any VxWorks platform is similar to building for a Linux target. To build a VxWorks application, create a makefile that reflects the compiler and linker for your target with appropriate flags defined. There will be several target-specific compile flags you must set to build correctly. For more information, see the *VxWorks Programmer's Guide*.

Table 8.2 Building Instructions for VxWorks Architectures

API	Library Format	Required RTI Libraries[ab]	Required Kernel Components	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnddscorez.a libnddscz.a libnddscppz.a or libnddscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	INCLUDE_TIMESTAMP INCLUDE_POSIX_CLOCKS For RTI architectures with SMP support also use: INCLUDE_TLS	-DRTI_VXWORKS When building with clang/llvm: also include -DRTI_VXWORKS_CLANG For all x64 platforms, also use: -DRTI_64BIT
	Static Debug	libnddscorezd.a libnddsczd.a libnddscppzd.a or libnddscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnddscore.so libnddsc.so libnddscpp.so or libnddscpp2.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so		
	Dynamic Debug	libnddscored.so libnddscd.so libnddscppd.so or libnddscpp2d.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so		

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The *Connext DDS* C/C++ libraries are in <NDDSHOME>/lib/<*architecture*>.

Table 8.2 Building Instructions for VxWorks Architectures

API	Library Format	Required RTI Libraries[ab]	Required Kernel Components	Required Compiler Flags
C	Static Release	libnddscorez.a libnddscz.a librticonnextmsgcz.a	INCLUDE_TIMESTAMP INCLUDE_POSIX_CLOCKS For RTI architectures with SMP support, also use: INCLUDE_TLS	-DRTI_VXWORKS For all x64 platforms, also use: -DRTI_64BIT
	Static Debug	libnddscorezd.a libnddsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnddscore.so libnddsc.so librticonnextmsgc.so		
	Dynamic Debug	libnddscored.so libnddscd.so librticonnextmsgcd.so		

Table 8.3 Running Instructions for VxWorks Architectures

RTI Architecture	Library Format (Release & Debug)	Environment Variables
VxWorks Kernel mode architectures	DKM	None required
VxWorks RTP architectures	Dynamic	LD_LIBRARY_PATH= <path_to_connext_libs>;<path_to_libc>" C
	Static	None required

[a]Choose *cpp*.* for the Traditional C++ API or *cpp2*.* for the Modern C++ API.

[b]The *Connext DDS* C/C++ libraries are in <NDDSHOME>/lib/<*architecture*>.

[c]In order to run dynamic RTP executables, you need to have the runtime libc.so library accessible. See the VxWorks Application Programmer's guide for more information.

Table 8.4 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv8Vx7SR0660llvm10.0.1.cortex-a53	Static Release	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_WRS_CONFIG_SMP -D_WRS_KERNEL -D__ELF__ -D__VXWORKS__ -D__vxworks -O --target=arm64 -ffixed-x18 -nostdlib -fno-omit-frame-pointer -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O -DNDEBUG -std=c11
	Static Debug	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O0 --target=arm64 -ffixed-x18 -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -mllvm -two-entry-phi-node-folding-threshold=2 -mno-implicit-float -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -g -std=c11
	Dynamic Release	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_WRS_CONFIG_SMP -D_WRS_KERNEL -D__ELF__ -D__VXWORKS__ -D__vxworks -O --target=arm64 -ffixed-x18 -nostdlib -fno-omit-frame-pointer -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DNDEBUG -std=c11
	Dynamic Debug	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\"armv8Vx7SR0660llvm10.0.1\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_WRS_CONFIG_SMP -D_WRS_KERNEL -D__ELF__ -D__VXWORKS__ -D__vxworks -O0 --target=arm64 -ffixed-x18 -nostdlib -fno-omit-frame-pointer -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration --target=arm64 -g -fdollars-in-identifiers -std=c11

Table 8.4 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp	Static Release	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O --target=arm64 -ffixed-x18 -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -mllvm -two-entry-phi-node-folding-threshold=2 -mno-implicit-float -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DNDEBUG -std=c11
	Static Debug	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O0 --target=arm64 -ffixed-x18 -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -mllvm -two-entry-phi-node-folding-threshold=2 -mno-implicit-float -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -g -std=c11
	Dynamic Release	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O --target=arm64 -ffixed-x18 -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -mllvm -two-entry-phi-node-folding-threshold=2 -mno-implicit-float -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DNDEBUG -fPIC -std=c11
	Dynamic Debug	-DARMEL -DCPU=_VX_CORTEX_A53 -DPtrIntType=long -DTARGET=\" armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -DVXWORKS_MAJOR_VERSION=7 -DVXWORKS_MINOR_VERSION=660 -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O0 --target=arm64 -ffixed-x18 -nostdlibinc -nostdinc++ -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -mllvm -two-entry-phi-node-folding-threshold=2 -mno-implicit-float -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -g -fPIC -std=c11

Table 8.4 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
pentium64Vx7.0gcc4.8.1	Release	-DCPU=_VX_NEHALEM -DPtrIntType=long -DTARGET-T=\"pentium64Vx7.0gcc4.8.1\" -DTOOL=gnu -DTOOL_FAMILY=gnu -D_WRS_KERNEL -O -march=atom -mpopcnt -nostdlib -fno-builtin -fno-defer-pop -m64 -fno-omit-frame-pointer -mcmodel=kernel -mno-red-zone -ansi -fno-implicit-fp -fno-zero-initialized-in-bss -Wall -Wno-unknown-pragmas -Wer-ror=implicit-function-declaration -O -DNDEBUG
	Debug	-DCPU=_VX_NEHALEM -DPtrIntType=long -DTARGET-T=\"pentium64Vx7.0gcc4.8.1\" -DTOOL=gnu -DTOOL_FAMILY=gnu -D_WRS_KERNEL -O0 -march=atom -mpopcnt -nostdlib -fno-builtin -fno-defer-pop -m64 -fno-omit-frame-pointer -mcmodel=kernel -mno-red-zone -ansi -fno-implicit-fp -fno-zero-initialized-in-bss -Wall -Wno-unknown-pragmas -Wer-ror=implicit-function-declaration -O0 -g
pentium64Vx7.0gcc4.8.1_rtp	Static Release	-DPtrIntType=long -DTARGET=\"pentium64Vx7.0gcc4.8.1_rtp\" -DTOOL_FAMILY=gnu -DVX_TOOL=gnu -D_C99 -D_HAS_C9X -D_SO64_SMALL__ -D_VX_CPU=_VX_NEHALEM -O -march=atom -mpopcnt -m64 -mc-model=small -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -MD -MP -std=c99 -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -O -DNDEBUG
	Static De-bug	-DPtrIntType=long -DTARGET=\"pentium64Vx7.0gcc4.8.1_rtp\" -DTOOL_FAMILY=gnu -DVX_TOOL=gnu -D_C99 -D_HAS_C9X -D_SO64_SMALL__ -D_VX_CPU=_VX_NEHALEM -O0 -march=atom -mpopcnt -m64 -mc-model=small -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -MD -MP -std=c99 -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -O0 -g
	Dynamic Release	-DPtrIntType=long -DTARGET=\"pentium64Vx7.0gcc4.8.1_rtp\" -DTOOL_FAMILY=gnu -DVX_TOOL=gnu -D_C99 -D_HAS_C9X -D_SO64_SMALL__ -D_VX_CPU=_VX_NEHALEM -D__SO_PICABILINUX__ -O -march=atom -mpopcnt -m64 -mcmodel=small -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -MD -MP -std=c99 -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -O -DNDEBUG -fPIC
	Dynamic Debug	-DPtrIntType=long -DTARGET=\"pentium64Vx7.0gcc4.8.1_rtp\" -DTOOL_FAMILY=gnu -DVX_TOOL=gnu -D_C99 -D_HAS_C9X -D_SO64_SMALL__ -D_VX_CPU=_VX_NEHALEM -D__SO_PICABILINUX__ -O0 -march=atom -mpopcnt -m64 -mcmodel=small -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -MD -MP -std=c99 -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -mrtp -O0 -g -fPIC

Table 8.4 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Vx21.11llvm12.0.1.1	Dynamic Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx21.11llvm12.0.1.1\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -D_WRS_CONFIG_SMP -D_WRS_KERNEL -DELF_ -D_VXWORKS_ -D__vx-works -O --target=x86_64-wrs-vxworks -m64 -march=core2 -mcmodel=kernel -mno-im-plicit-float -mno-red-zone -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -ftls-model=local-exec -nostdlibinc -nostdinc++ -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DNDEBUG -std=c11
	Dynamic Debug	-DCPU=VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx21.11llvm12.0.1.1\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -D_WRS_CONFIG_SMP -D_WRS_KERNEL -DELF_ -D_VXWORKS_ -D__vx-works -O0 --target=x86_64-wrs-vxworks -m64 -march=core2 -mcmodel=kernel -mno-im-plicit-float -mno-red-zone -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -ftls-model=local-exec -nostdlibinc -nostdinc++ -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -g -std-d=c11

Table 8.4 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Vx21.11llvm12.0.1.1_rtp	Static Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx21.11llvm12.0.1.1_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -DELF_ -D_RTP_ -D_VXWORKS_ -D__vxworks -O --target=x86_64-wrs-vx-works -m64 -march=core2 -mcmodel=small -fasm -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -nostdlibinc -nostdinc++ -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DNDEBUG -std=c11
	Static Debug	-DCPU=VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx21.11llvm12.0.1.1_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -DELF_ -D_RTP_ -D_VXWORKS_ -D__vxworks -O0 --target=x86_64-wrs-vx-works -m64 -march=core2 -mcmodel=small -fasm -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -nostdlibinc -nostdinc++ -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -g -std-d=c11
	Dynamic Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx21.11llvm12.0.1.1_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -DELF_ -D_RTP_ -D_VXWORKS_ -D__vxworks -O --target=x86_64-wrs-vx-works -m64 -march=core2 -mcmodel=small -fasm -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -nostdlibinc -nostdinc++ -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -DNDEBUG -fPIC -std=c11
	Dynamic Debug	-DCPU=VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx21.11llvm12.0.1.1_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -DELF_ -D_RTP_ -D_VXWORKS_ -D__vxworks -O0 --target=x86_64-wrs-vx-works -m64 -march=core2 -mcmodel=small -fasm -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -nostdlibinc -nostdinc++ -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -g -fPIC -std=c11

Table 8.4 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Vx7SR0630llvm8.0.0.2	Release	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx7SR0630llvm8.0.0.2\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_WRS_CONFIG_SMP -D_WRS_KERNEL -D__ELF__ -D__VXWORKS__ -D__vxworks -O --target=x86_64 -m64 -mcmodel=kernel -mno-red-zone -nostdlib -fno-omit-frame-pointer -march=core2 -nostdlibinc -nostdinc++ -mno-implicit-float -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O -DNDEBUG -std=c11
	Debug	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx7SR0630llvm8.0.0.2\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_WRS_CONFIG_SMP -D_WRS_KERNEL -D__ELF__ -D__VXWORKS__ -D__vxworks -O0 --target=x86_64 -m64 -mcmodel=kernel -mno-red-zone -nostdlib -fno-omit-frame-pointer -march=core2 -nostdlibinc -nostdinc++ -mno-implicit-float -ftls-model=local-exec -fno-builtin -fno-strict-aliasing -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O0 -g -std=c11
x64Vx7SR0630llvm8.0.0.2_rtp	Static Release	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx7SR0630llvm8.0.0.2_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O --target=x86_64 -m64 -mcmodel=small -fno-omit-frame-pointer -march=core2 -fno-strict-aliasing -fno-builtin -nostdlibinc -nostdinc++ -fasm -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O -DNDEBUG -std=c11
	Static Debug	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx7SR0630llvm8.0.0.2_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O0 --target=x86_64 -m64 -mcmodel=small -fno-omit-frame-pointer -march=core2 -fno-strict-aliasing -fno-builtin -nostdlibinc -nostdinc++ -fasm -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O0 -g -std=c11
	Dynamic Release	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx7SR0630llvm8.0.0.2_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O --target=x86_64 -m64 -mcmodel=small -fno-omit-frame-pointer -march=core2 -fno-strict-aliasing -fno-builtin -nostdlibinc -nostdinc++ -fasm -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O -DNDEBUG -fPIE -std=c11
	Dynamic Debug	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET-T=\"x64Vx7SR0630llvm8.0.0.2_rtp\" -DTOOL=llvm -DTOOL_FAMILY=llvm -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D__ELF__ -D__RTP__ -D__VXWORKS__ -D__vxworks -O0 --target=x86_64 -m64 -mcmodel=small -fno-omit-frame-pointer -march=core2 -fno-strict-aliasing -fno-builtin -nostdlibinc -nostdinc++ -fasm -Wall -Wno-unknown-pragmas -Werror=implicit-function-declaration -O0 -g -fPIE -std=c11

8.1 Notes for VxWorks 7 Platforms

- Required Makefile Change

  For VxWorks 7.0 platforms only: After you run *rtiddsgen*, either edit the generated makefile to specify which VxWorks Source Build (VSB) you want to use or set an environment variable called VSB_DIR that points to the VSB. In the generated makefile, find this line and change it to match your VSB directory:

  ```
  VSB_DIR = # Specify your VSB directory here.
  ```

  Note: RTI uses a VSB based on the itl_generic BSP (provided by Wind River) to build the *Connext DDS* libraries for VxWorks 7.0 x64.
- For VxWorks 7 SR0600 and newer releases:

  To run VxWorks tasks with Thread Local Storage, the kernel must be configured in advance with an explicit size for the TLS variables through the kernel parameter, DKM_TLS_SIZE. To run *Connext DDS* in a VxWorks task, DKM_TLS_SIZE must be 160 or higher to fit the TLS variables. For more information, see the **tlsLib** API reference in your VxWorks 7 documentation.
- To avoid symbol duplication in applications generated with *rtiddsgen*, in statically linked Downloadable Kernel Modules (DKMs):

  When using rtiddsgen to generate a Connext DDS application, publisher and subscriber are created. By default, the generated makefile will create a separate application for the publisher and the subscriber. This poses a problem when linking static kernel modules. In this case, you would have a static DKM containing the publisher application + Connext DDS libraries, and another static DKM containing the subscriber application + Connext DDS libraries. When those two modules are loaded into the kernel, all the Connext DDS symbols will be duplicated and you will likely run into issues.

  To overcome this limitation, an additional target is created in the Makefile for the VxWorks kernel architectures called **pubsub**. This target will create a single DKM containing both the publisher and subscriber application, plus the Connext DDS libraries. With this approach, you can link this single DKM and still have the publisher and subscriber applications available in the kernel without duplication of symbols.

8.2 Known Defects

- When using VxWorks 7.0 64-bit RTP mode, there is a bug in the **getsockopt()** function: the **optlen** parameter is not properly set. Refer to Wind River defect V7NET-1293
- When using VxWorks 7.0 64-bit RTP mode, an incorrect number of sections is introduced in the resulting ELF binaries, so the VxWorks kernel cannot load them. Refer to Wind River defect VXW7-3771.
- When using VxWorks 6 or 7, a too-small SO_RCVBUF causes packet loss in polled receiving in VxWorks SMP

Due to a potential bug in the VxWorks Network stack (V7NET-2540), creating a socket with a very small SO_RCVBUF might cause packet loss when receiving over that socket in a non-blocking way (polling). This problem has been reproduced in SMP kernels.

To work around this, create the receiving sockets with a SO_RCVBUF size of at least 4000. This is twice the minimum size allowed by VxWorks (IPNET_MIN_RCVBUF_SIZE, which is 2000).

- There is a known issue (Wind River Defect ID V7PRO-6555) with VxWorks kernels where clock_gettime may take more time than expected on Intel boards. This can happen when setting the kernel parameter "HIGH_RES_POSIX_CLOCK" to TRUE (FALSE by default). We recommend leaving that parameter set to the default value (FALSE).

8.3 Increasing the Stack Size

Connext DDS applications may require more than the default stack size on VxWorks.

To prevent stack overrun, you can create/enable the *DomainParticipant* in a thread with a larger stack, or increase the default stack size of the shell task by recompiling the kernel. For more information, please see the Solutions on the RTI Community portal, accessible from https://community.rti.com/kb.

8.4 Enabling Floating Point Coprocessor in Kernel Tasks

Some applications may require you to spawn the kernel with floating-point coprocessor support. To do so, you must pass the VX_FP_TASK option to the "options" argument of taskSpawn (please refer to Wind River documentation for more information about taskSpawn arguments).

If you spawn the task from the c-shell, the VX_FP_TASK definition is not available and you must provide a numeric value: 0x1000000 for VxWorks 6.x and newer versions. If the target system runs a PowerPC e500v2 CPU, you need to pass VX_SPE_TASK instead, whose value is 0x4000000.

8.5 Downloadable Kernel Modules (DKM) for Kernel Mode on VxWorks Systems

The *Connext DDS* Professional, Research, and Evaluation packages include support for the Request-Reply Communication Pattern, for all platforms in Table 8.1 VxWorks Target Platforms on page 89 and all programming languages, with one exception: VxWorks 6.9.4.6 supports Request-Reply in the C API only.

In VxWorks kernel mode, dynamic libraries are not supported. Instead, Downloadable Kernel Modules (DKMs) are used. Once a DKM has been loaded into the kernel, all the symbols from that DKM will be accessible from the kernel.

In VxWorks kernel mode, before a C++ DKM can be downloaded to the VxWorks kernel, it must undergo an additional host processing step known as *munching*. This step is necessary for proper initialization of static objects and to ensure that the C++ run-time support calls the correct

constructor/destructors in the correct order for all static objects. All the *Connext DDS* DKMs (**libnddscore.so**, **libnddsc.so**, **libnddscpp.so**, etc) are shipped already munched.

When you create an application as a DKM for use in kernel mode, you have two options for linking:

- Perform a static linkage: This involves linking all the needed *Connext DDS* libraries inside the DKM (such as **libnddscorez.a**). Note that if you plan to load several statically linked DKMs into the kernel, you will have issues related to duplicate symbols, because the symbols from *Connext DDS* will be loaded once per DKM.
- Perform a partial linkage: This involves building your application without linking against the *Connext DDS* libraries. Later, at load time, you will need to load into the kernel the required *Connext DDS* libraries and your application DKM. This is recommended if you plan to have more than one DKM using *Connext DDS*.

For both options, you will need to munch your application DKMs.

8.6 Libraries for RTP Mode on VxWorks Systems

Dynamic libraries are *not* available for VxWorks systems with Real Time Processes (RTP mode) on PowerPC (PPC) CPUs. This is due to a platform limitation in VxWorks PPC platforms that puts an upper bound on the size of the Global Offset Table (GOT) for any single library, which limits how many symbols the library can export. Some *Connext DDS* libraries (in particular, libnddsc) export a number of symbols that exceed this upper bound.

Dynamic libraries *are* available for VxWorks systems with RTP mode on Intel and Arm CPUs.

8.7 Requirement for Restarting Applications

When restarting a VxWorks application, you may need to change the 'appId' value. In general, this is only required if you still have other *Connext DDS* applications running on other systems that were talking to the restarted application. If all the *Connext DDS* applications are restarted, there should be no problem.

This section explains why this is necessary and how to change the appId.

All *Connext DDS* applications must have a unique GUID (globally unique ID). This GUID is composed of a hostId and an appId. RTI implements unique appIds by using the process ID of the application. On VxWorks systems, an application's process ID will often be the same across reboots. This may cause logged errors during the discovery process, or discovery may not complete successfully for the restarted application.

The workaround is to manually provide a unique appId each time the application starts. The appId is stored in the *DomainParticipant's* WireProtocol QosPolicy. There are two general approaches to providing a unique appId. The first approach is to save the appId in NVRAM or the file system, and then incre-

ment the appId across reboots. The second approach is to base the appId on something that is likely to be different across reboots, such as a time-based register.

8.8 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs. The Modern C++ API is available for all supported VxWorks platforms.

These platforms have only been tested with C++11:

- VxWorks 7.0 SR0510: pentium64Vx7.0gcc4.8.1_rtp
- VxWorks 7.0 SR0630: x64Vx7SR0630llvm8.0.0.2_rtp
- VxWorks 7.0 SR0660: armv8Vx7SR0660llvm10.0.1.cortex-a53 and armv8Vx7SR0660llvm10.0.1.cortex-a53_rtp
- VxWorks 21.11: x64Vx21.11llvm12.0.1.1and x64Vx21.11llvm12.0.1.1_rtp

These platforms have only been tested with C++03:

- VxWorks 7.0 SR0630: x64Vx7SR0630llvm8.0.0.2
- VxWorks 7.0 SR0510: pentium64Vx7.0gcc4.8.1_rtp

Notes:

- Support for C++03 is deprecated starting with release 6.1.0, which is the last release that supports non-C++11-compliant compilers. After release 6.1.0, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator*starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

8.8.1 How to build VxWorks applications that use both Boost and the Connext DDS Modern C++ API

The Modern C++ API internally uses a subset of Boost 1.61. All the Boost symbols have been renamed to avoid collisions with user applications that also include Boost. However, some standard functions that are missing from VxWorks are defined in Boost headers as inline functions (symlink, readlink, times, truncate). Source files that include Boost and the *Connext DDS* Modern C++ API may fail to compile due to

duplicate symbols, because these functions are defined both in the Boost headers used by the Modern C++ API and the user-included headers.

To avoid these errors, you will need to perform one of the following options:

- Make sure the Boost headers are included *before* any RTI header. For example:

  ```
  #include <boost/shared_ptr.hpp> // FIRST
  ...
  #include <dds/domain/DomainParticipant.hpp> // SECOND
  ...
  ```

  The RTI Boost headers will detect that another Boost installation has been included, and will exclude the conflicting symbols.

- Compile the source files that use Boost with the option **-DRTI_USE_BOOST**. The RTI Boost headers will recognize this preprocessor definition and exclude the conflicting symbols.

8.9 Multicast Support

Multicast is supported on all VxWorks architectures.

It is configured out of the box. That is, the default value for the initial peers list (NDDS_DISCOVERY_PEERS) includes a multicast address. See the API Reference HTML documentation for more information.

Known Defects related to multicast:

- If you have a Wind River account, you can find more information about defect VXW6-19089 (also known as WIND00418701) here: https://support2.windriver.com/index.php?page=defects&on=view&id=VXW6-19089.

  If you need a patch for your version of VxWorks, or for more information about this issue, please contact Wind River.

8.10 Supported Transports

- **Shared memory:** Shared memory is supported and enabled by default on all VxWorks 6.x and higher architectures. See also:
  - 8.10.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID on the next page
  - 8.10.2 How To Run Connext DDS Libraries in Kernels Built without Shared Memory on the next page
- **UDPv4:** Supported and enabled by default.

- **UDPv6:** Supported on VxWorks 6.7 and higher platforms, except VxWorks 6.9.4.6. No Traffic Class support.
- **TCP/IPv4:** Not supported.

8.10.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID

By default, applications using the auto-generated Participant ID (-1) cannot communicate between user space and kernel space on the same host via SHMEM. The root cause is that the participants use the same participant ID. Therefore the workaround for this issue is to explicitly provide a participant ID when creating the *DomainParticipants*. The participant ID is set in the *DomainParticipant's* WireProtocol QoS policy.

8.10.2 How To Run Connext DDS Libraries in Kernels Built without Shared Memory

Since *Connext DDS* libraries support shared memory as a built-in transport, building a kernel without shared-memory support will cause loading or linking errors, depending on whether the *Connext DDS* libraries are loaded after boot, or linked at kernel build time.

The most straightforward way to fix these errors is to include shared-memory support in the kernel (INCLUDE_SHARED_DATA in the kernel build parameters).

However, in some versions of VxWorks, it is not possible to include shared-memory support without also including RTP support. If you are unwilling or unable to include shared-memory support in your configuration, you will need to do the following:

1. Add the component INCLUDE_POSIX_SEM
2. Define stubs that return failure for the missing symbols **sdOpen** and **sdUnmap** as described below:
   - For **sdOpen**, we recommend providing an implementation that returns NULL, and sets errno to ENOSYS. For the function prototype, refer to the file **sdLib.h** in the VxWorks distribution.
   - For **sdUnmap**, we recommend providing an implementation that returns ERROR and sets errno to ENOSYS. For the function prototype, refer to the file **sdLibCommon.h** in the VxWorks distribution.

In addition to providing the symbol stubs for **sdOpen** and **sdUnmap**, we also recommend disabling the SHMEM transport by using the **transport_builtin** mask in the QoS configuration.

8.11 Unsupported Features

The Modern C++ API for Request-Reply Communication is not supported on VxWorks 6.9.4.6 platforms. The Traditional C++ API for Request-Reply is supported.

These features are not supported on any VxWorks platforms:

- Backtrace
- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script
- Persistence Service
- Dynamic loading of *RTI Security Plugins* in kernel mode on VxWorks 7.0 (SR0630) platforms
- Remote Procedure Calls on x64 platforms

8.12 Monotonic Clock Support

The monotonic clock (described in *Clock Selection,* in the *Domains* chapter of the RTI Connext DDS Core Libraries User's Manual) is supported on VxWorks 6.x and higher platforms.

8.13 Use of Real-Time Clock

Starting with 5.3.0, *Connext DDS* uses the Real Time Clock to get the time from the System Clock on VxWorks 6.x and higher platforms. Previously **tickGet()** was used for the system clock.

8.14 Thread Configuration

See these tables:

- Table 8.5 Thread Setting for VxWorks Platforms (Applies to Kernel Tasks or Real-Time Process Threads) on the next page
- Table 8.6 Thread-Priority Definitions for VxWorks Platforms on page 107
- Table 8.7 Thread Kinds for VxWorks Platforms on page 107

Table 8.5 Thread Setting for VxWorks Platforms (Applies to Kernel Tasks or Real-Time Process Threads)

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	100
	stack_size	30 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	120
	stack_size	30 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	110
	stack_size	4 * 30 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	71
	stack_size	4 * 30 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 8.6 Thread-Priority Definitions for VxWorks Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	100
THREAD_PRIORITY_HIGH	68
THREAD_PRIORITY_ABOVE_NORMAL	71
THREAD_PRIORITY_NORMAL	100
THREAD_PRIORITY_BELOW_NORMAL	110
THREAD_PRIORITY_LOW	120

Table 8.7 Thread Kinds for VxWorks Platforms

Thread Kinds	Operating-System Configuration[a]
DDS_THREAD_SETTINGS_FLOATING_POINT	Uses VX_FP_TASK when calling taskSpawn()
DDS_THREAD_SETTINGS_STDIO	Uses VX_STDIO when calling taskSpawn() (Kernel mode only)
DDS_THREAD_SETTINGS_REALTIME_PRIORITY	Configures the schedule policy to SCHED_FIFO.
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE	N/A

8.15 Libraries Required for Using Distributed Logger

RTI Distributed Logger is supported all VxWorks architectures.

Table 8.8 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

[a]See VxWorks manuals for more information.

Table 8.8 Additional Libraries for using RTI Distributed Logger

Language	Static[a]		Dynamic[b]	
	Release	Debug	Release	Debug
C	librtidlcz.a	librtidlczd.a	librtidlc.so	librtidcd.so
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a	librtidlc.so librtidlcpp.so	librtidlcd.so librtidlcppd.so

8.16 Libraries Required for Using Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the static release version of the *Connext DDS* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

If you are statically linking your application with DDS libraries and you want to add monitoring to your application, you will also need to statically link the monitoring library. The library cannot be loaded dynamically strictly through the QoS profile because it also depends on DDS to publish its data. Therefore, it depends on DDS; the DDS functionality would cause duplicate symbols to be found resulting, in the termination of the process.

Notes:

- Automatic loading of the dynamic monitoring library through QoS is not supported.
- Memory and CPU usage is not available in monitoring data.
- If you plan to use *static* libraries, the RTI library from Table 8.9 Additional Libraries for Using Monitoring must appear *first* in the list of libraries to be linked.

Table 8.9 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[c]
Dynamic Release	librtimonitoring.so[d]

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]These libraries are in <NDDSHOME>/lib/<architecture>.

[c]These libraries are in <NDDSHOME>/lib/<architecture>.

[d]Dynamic libraries are not supported for VxWorks platforms on PPC CPUs using RTP mode.

Table 8.9 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Dynamic Debug	librtimonitoringd.so[b]
Static Release	librtimonitoringz.a
Static Debug	librtimonitoringzd.a

8.17 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the RTI Real-Time WAN Transport Installation Guide for details.

Using *Real-Time WAN Transport* requires one of the libraries in Table 8.10 Additional Libraries for Using RTI Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

Table 8.10 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[c]
Dynamic Release	libnddsrwt.so
Dynamic Debug	libnddsrwtd.so
Static Release	libnddsrwtz.a
Static Debug	libnddsrwtzd.a

8.18 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, link against the additional library in Table 8.11 Additional Libraries for Zero Copy Transfer Over Shared Memory.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

[b]Dynamic libraries are not supported for VxWorks platforms on PPC CPUs using RTP mode.

[c]These libraries are in <NDDSHOME>/lib/<architecture>.

Table 8.11 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Over Shared Memory Libraries[a]
Dynamic Release	libnddsmetp.so
Dynamic Debug	libnddsmetpd.so
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

8.19 Increasing the Receive Socket Buffer Size

For *Connext DDS* applications running on VxWorks 6.7 or higher systems and using UDPv4, we recommend setting the property **dds.transport.UDPv4.builtin.recv_socket_buffer_size** to a value of 128000 or higher. This recommendation is due to Wind River's usage of extra receive socket buffer space to correct Wind River defect number WIND00135312.

[a]These libraries are in <NDDSHOME>/lib/<architecture>.

Chapter 9 Windows Platforms

First, see the basic instructions for compiling Windows Platforms in the *Building Applications* chapter in the RTI Connext DDS Core Libraries User's Manual.

The following tables provide supplemental information. Table 9.1 Supported Windows Platforms lists the architectures supported on Windows operating systems.

Table 9.1 Supported Windows Platforms

Operating System	Visual Studio® Version	RTI Architecture Abbreviation	.NET Version[a]	JDK Version	CPU
Windows 10	VS 2015 Update 3	x64Win64VS2015	4.6	JDK 11	x64
	VS 2017 Update 2 VS 2019 Version 16.0.0 VS 2022 Version 17.0	x64Win64VS2017	4.6.1		
Windows 11	VS 2017 Update 2 VS 2019 Version 16.0.0 VS 2022 Version 17.0	x64Win64VS2017	4.6.1		
Windows Server 2012 R2	VS 2015 Update 3	x64Win64VS2015	4.6		
Windows Server 2016	VS 2015 Update 3	x64Win64VS2015	4.6		
	VS 2017 Update 2 VS 2019 Version 16.0.0 VS 2022 Version 17.0	x64Win64VS2017	4.6.1		

Note regarding .NET API Support: The .NET API is supported on Windows 10, but it doesn't support Visual Studio 2012-2015 for development. Development is supported in Visual Studio 2017-2019, Visual Studio Code, and the .NET CLI.

The compiler flags and the libraries you will need to link into your application are in the following tables:

- Table 9.2 Building Instructions for Windows Host Architectures
- Table 9.3 Building Instructions for Windows Target Architectures

See also:

- 9.15 Libraries Required for Using Distributed Logger Support on page 128
- 9.16 Libraries Required for Using Monitoring on page 129
- 9.17 Libraries Required for Using RTI Real-Time WAN Transport APIs on page 130

[a]The RTI .NET assemblies are supported for both the C++/CLI and C# languages. The type support code generated by rtiddsgen is in C++/CLI; compiling the generated type support code requires Microsoft Visual C++. Calling the assembly from C# requires Microsoft Visual C#.

- 9.18 Libraries Required for Using RTI Secure WAN Transport APIs on page 130
- 9.19 Libraries Required for Using RTI TCP Transport APIs on page 131
- 9.20 Libraries Required for Zero Copy Transfer Over Shared Memory on page 132

Make sure you are consistent in your use of static (.lib), dynamic (.dll), debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 9.4 Running Instructions for Windows Architectures on page 118 provides details on the environment variables that must be set at run time for a Windows architecture.

For details on how the libraries were built by RTI, see Table 9.5 Library-Creation Details for Windows Architectures. This information is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

Table 9.2 Building Instructions for Windows Host Architectures

API	Library Format	RTI Libraries or Jar Files [a] [b] [c]	Required System Libraries	Required Compiler Flags
C	Static Release	nddscorez.lib nddscz.lib rticonnextmsgcz.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	/D "RTI_WIN32" /MD
	Static Debug	nddscorezd.lib nddsczd.lib rticonnextmsgczd.lib		/D "RTI_WIN32" /MDd
	Dynamic Release	nddscore.lib nddsc.lib rticonnextmsgc.lib		/D "RTI_WIN32" /D "NDDS_DLL_VARIABLE" /MD
	Dynamic Debug	nddscored.lib nddscd.lib rticonnextmsgcd.lib		/D "RTI_WIN32" /D "NDDS_DLL_VARIABLE" /MDd

[a] Choose ***cpp*.*** for the Traditional C++ API or ***cpp2*.*** for the Modern C++ API.

[b] The RTI C/C++/Java libraries are in **<NDDSHOME>\lib\<*architecture*>**. Jar files are in <NDDSHOME>\lib\java.

[c] For C++/CLI and C#: The **nddsdotnet** and **rticonnextmsgdotnet** library names include a *<version>*, which depends on your version of .NET. Use the digits, such as 451 or 46. See Table 9.1 Supported Windows Platforms on the previous page for supported .NET versions.

Table 9.2 Building Instructions for Windows Host Architectures

API	Library Format	RTI Libraries or Jar Files [a] [b] [c]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	nddscorez.lib nddscz.lib nddscppz.lib or nddscpp2z.lib rticonnextmsgcppz.lib or rticonnextmsgcpp2z.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	/D "RTI_WIN32" /MD
	Static Debug	nddscorezd.lib nddsczd.lib nddscppzd.lib or nddscpp2zd.lib rticonnextmsgcppzd.lib or rticonnextmsgcpp2zd.lib		/D "RTI_WIN32" /MDd
	Dynamic Release	nddscore.lib nddsc.lib nddscpp.lib or nddscpp2.lib rticonnextmsgcpp.lib or rticonnextmsgcpp2.lib		/D "RTI_WIN32" /D "NDDS_DLL_ VARIABLE" /MD
	Dynamic Debug	nddscored.lib nddscd.lib nddscppd.lib or nddscpp2d.lib rticonnextmsgcppd.lib or rticonnextmsgcpp2d.lib		/D "RTI_WIN32" /D "NDDS_DLL_ VARIABLE" /MDd

[a] Choose ***cpp*.*** for the Traditional C++ API or ***cpp2*.*** for the Modern C++ API.

[b] The RTI C/C++/Java libraries are in **<NDDSHOME>\lib\<*architecture*>**. Jar files are in <NDDSHOME>\lib\java.

[c] For C++/CLI and C#: The **nddsdotnet** and **rticonnextmsgdotnet** library names include a *<version>*, which depends on your version of .NET. Use the digits, such as 451 or 46. See Table 9.1 Supported Windows Platforms on the previous page for supported .NET versions.

Table 9.2 Building Instructions for Windows Host Architectures

API	Library Format	RTI Libraries or Jar Files [a] [b] [c]	Required System Libraries	Required Compiler Flags
C++/CLI	Release	nddscore.lib nddsc.lib nddscpp.lib nddsdotnet<*version*>.dll rticonnextmsgdotnet<*version*>.dll	N/A	/D "RTI_WIN32" /D "NDDS_DLL_VARIABLE" /MD /D "WIN32_LEAN_AND_MEAN"
	Debug	nddscored.lib nddscd.lib nddscppd.lib nddsdotnet<*version*>d.dll rticonnextmsgdotnet<*version*>d.dll		/D "RTI_WIN32" /D "NDDS_DLL_VARIABLE" /MDd /D "WIN32_LEAN_AND_MEAN"
C#	Release	nddsdotnet<*version*>.dll rticonnextmsgdotnet<*version*>.dll	N/A	N/A
	Debug	nddsdotnet<*version*>d.dll rticonnextmsgdotnet<*version*>d.dll		
Java	Release	nddsjava.jar rticonnextmsg.jar	N/A	N/A
	Debug	nddsjavad.jar rticonnextmsgd.jar		

[a] Choose ***cpp*.*** for the Traditional C++ API or ***cpp2*.*** for the Modern C++ API.

[b] The RTI C/C++/Java libraries are in **<NDDSHOME>\lib\<*architecture*>**. Jar files are in <NDDSHOME>\lib\java.

[c] For C++/CLI and C#: The **nddsdotnet** and **rticonnextmsgdotnet** library names include a <*version*>, which depends on your version of .NET. Use the digits, such as 451 or 46. See Table 9.1 Supported Windows Platforms on the previous page for supported .NET versions.

Table 9.3 Building Instructions for Windows Target Architectures

API	Library Format	RTI Libraries or Jar Files [a] [b]	Required System Libraries	Required Compiler Flags
C	Static Release	nddscorez.lib nddscz.lib rticonnextmsgcz.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	/Gd /MD /D "WIN32" /D "RTI_WIN32" /D "NDEBUG"
	Static Debug	nddscorezd.lib nddsczd.lib rticonnextmsgczd.lib		/Gd /MDd /D "WIN32" /D "RTI_WIN32"
	Dynamic Release	nddscore.lib nddsc.lib rticonnextmsgc.lib		/Gd /MD /D "WIN32" /D "NDDS_DLL_ VARIABLE" /D "RTI_WIN32" /D "NDEBUG"
	Dynamic Debug	nddscored.lib nddscd.lib rticonnextmsgcd.lib		/Gd /MDd /D "WIN32" /D "NDDS_DLL_ VARIABLE" /D "RTI_WIN32"

[a]The RTI C/C++/Java libraries are in <NDDSHOME>\lib\<*architecture*>. Jar files are in <NDDSHOME>\lib\java.

[b]For C++/CLI and C#: The **nddsdotnet** and **rticonnextmsgdotnet** library names include a <*version*>, which depends on your version of .NET. Use the digits, such as 451 or 46. See Table 9.1 Supported Windows Platforms on page 112 for supported .NET versions.

Table 9.3 Building Instructions for Windows Target Architectures

API	Library Format	RTI Libraries or Jar Files [a] [b]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	nddscorez.lib nddscz.lib nddscppz.lib or nddscpp2z.lib rticonnextmsgcppz.lib or rticonnextmsgcpp2z.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	/Gd /EHsc /MD /D "WIN32" /D "RTI_WIN32" /D "NDEBUG"
	Static Debug	nddscorezd.lib nddsczd.lib nddscppzd.lib or nddscpp2zd.lib rticonnextmsgcppzd.lib or rticonnextmsgcpp2zd.lib		/Gd /EHsc /MDd /D "WIN32" /D "RTI_WIN32"
	Dynamic Release	nddscore.lib nddsc.lib nddscpp.lib or nddscpp2.lib rticonnextmsgcpp.lib or rticonnextmsgcpp2.lib		/Gd /EHsc /MD /D "WIN32" /D "NDDS_DLL_ VARIABLE" /D "RTI_WIN32" /D "NDEBUG"
	Dynamic Debug	nddscored.lib nddscd.lib nddscppd.lib or nddscpp2d.lib rticonnextmsgcppd.lib or rticonnextmsgcpp2d.lib		/Gd /EHsc /MDd /D "WIN32" /D "NDDS_DLL_ VARIABLE" /D "RTI_WIN32"

[a]The RTI C/C++/Java libraries are in <NDDSHOME>\lib\<*architecture*>. Jar files are in <NDDSHOME>\lib\java.

[b]For C++/CLI and C#: The **nddsdotnet** and **rticonnextmsgdotnet** library names include a <*version*>, which depends on your version of .NET. Use the digits, such as 451 or 46. See Table 9.1 Supported Windows Platforms on page 112 for supported .NET versions.

Table 9.3 Building Instructions for Windows Target Architectures

API	Library Format	RTI Libraries or Jar Files [a] [b]	Required System Libraries	Required Compiler Flags
C#	Release	nddsdotnet<*version*>.dll rticonnextmsgdotnet<*version*>.dll	N/A	N/A
	Debug	nddsdotnet<*version*>d.dll rticonnextmsgdotnet<*version*>d.dll		
C++/CLI	Release	nddscore.lib nddsc.lib nddscpp.lib rticonnextmsgdotnet<*version*>.dll	netapi32.lib advapi32.lib user32.lib ws2_32.lib	/Gd /EHsc /MD /D "WIN32" /D "NDDS_DLL_ VARIABLE" /D "RTI_WIN32" /D "NDEBUG"
	Debug	nddscored.lib nddscd.lib nddscppd.lib rticonnextmsgdotnet<*version*>d.dll		/Gd /EHsc /MDd /D "WIN32" /D "NDDS_DLL_ VARIABLE" /D "RTI_WIN32"
Java	Release	nddsjava.jar rticonnextmsg.jar	N/A	N/A
	Debug	nddsjavad.jar rticonnextmsgd.jar		

Table 9.4 Running Instructions for Windows Architectures

RTI Architecture	Library Format	Environment Variables[c]
All supported Windows architectures for Java	N/A	Path=%NDDSHOME%\lib\<*architecture*>; %Path%
All other supported Windows architectures	Static (Release and Debug)	None required
	Dynamic (Release and Debug)	Path=%NDDSHOME%\lib\<*architecture*>; %Path%

[a]The RTI C/C++/Java libraries are in <NDDSHOME>\lib\<*architecture*>. Jar files are in <NDDSHOME>\lib\java.

[b]For C++/CLI and C#: The **nddsdotnet** and **rticonnextmsgdotnet** library names include a <*version*>, which depends on your version of .NET. Use the digits, such as 451 or 46. See Table 9.1 Supported Windows Platforms on page 112 for supported .NET versions.

[c]**%Path%** represents the value of the **Path** variable prior to changing it to support *Connext DDS*. When using **nddsjava.jar**, the Java virtual machine (JVM) will attempt to load release versions of the native libraries. When using **nddsjavad.jar**, the JVM will attempt to load debug versions of the native libraries.

Table 9.5 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Win64VS2012 Note: linker requires /MACHINE:X64 option.	Static Release	/W3 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
x64Win64VS2013 Note: linker requires /MACHINE:X64 option.	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c

Table 9.5 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Win64VS2015 Note: linker requires /MACHINE:X64 option.	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
x64Win64VS2017 Note: linker requires /MACHINE:X64 option.	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET=\"x64Win64VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
All 64-bit Windows architectures for .NET	Dynamic Release	/O2 /GL /D "WIN64" /D "NDEBUG" /D "NDDS_DLL_VARIABLE" /D "_WINDLL" /D "_UNICODE" /D "UNICODE" /FD /EHa /MD /c /Zi /clr /TP
	Dynamic Debug	/Od /D "WIN64" /D "_DEBUG" /D "NDDS_DLL_VARIABLE" /D "_WINDLL" /D "_UNICODE" /D "UNICODE" /FD /EHa /MDd /c /Zi /clr /TP
All 64-bit Windows architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

Table 9.5 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
i86Win32VS2012 (deprecated, available on demand)	Static Release	-DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"i86Win32VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	-DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"i86Win32VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	-DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"i86Win32VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_ DEPRECATE -c
	Dynamic Debug	-DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGE T=\"i86Win32VS2012\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
i86Win32VS2013 (deprecated, available on demand)	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"i86Win32VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"i86Win32VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"i86Win32VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGE T=\"i86Win32VS2013\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c

Table 9.5 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
i86Win32VS2015 (deprecated, available on demand)	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DSTDC99 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DSTDC99 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DSTDC99 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2015\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
i86Win32VS2017 (deprecated, available on demand)	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATE -DNDEBUG -c
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmtd.lib" /defaultlib:"msvcrtd.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=I80586 -DTARGET=\"x86Win32VS2017\" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0501 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATE -c
All 32-bit Windows architectures for .NET (deprecated, available on demand)	Dynamic Release	/O2 /Oy- /GL /D "WIN32" /D "NDEBUG" /D "NDDS_DLL_VARIABLE" /D "_WINDLL" /D "_UNICODE" /D "UNICODE" /FD /EHa /MD /c /Zi /clr /TP
	Dynamic Debug	/Od /D "WIN32" /D "_DEBUG" /D "NDDS_DLL_VARIABLE" /D "_WINDLL" /D "_UNICODE" /D "UNICODE" /FD /EHa /MDd /c /Zi /clr /TP

9.1 Requirements when Using Microsoft Visual Studio

Note: Debug versions of applications and the various Visual C++ DLLs are not redistributable. Therefore, if you want to run debug versions, you must have the compiler installed.

When Using Visual Studio 2015 — Update 3 Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2015 Update 3 installed on the machine

where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2015 Update 3 from this Microsoft website: https://www.microsoft.com/en-us/download/details.aspx?id=53840.

When Using Visual Studio 2017 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2017 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2017 from this Microsoft website: https://visualstudio.microsoft.com/vs/older-downloads/. Then look in this section: "Redistributables and Build Tools" for "Microsoft Visual C++ Redistributable for Visual Studio 2017".

When Using Visual Studio 2019 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2019 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2019 from this Microsoft website: https://www.visualstudio.com/downloads/. Then look in this section: "Other Tools and Frameworks" for "Microsoft Visual C++ Redistributable for Visual Studio 2019".

When Using Visual Studio 2022 version 17 — Redistributable Package Requirement

You must have the Visual C++ Redistributable for Visual Studio 2022 installed on the machine where you are running an application linked with dynamic libraries. This includes C/C++ dynamically linked and all .NET and Java applications.

You can download the Visual C++ Redistributable for Visual Studio 2022 from this Microsoft website: https://www.visualstudio.com/downloads/. Then look in this section: "Other Tools and Frameworks" for "Microsoft Visual C++ Redistributable for Visual Studio 2022".

9.2 Linking with Libraries for Windows Platforms

Starting with *Connext DDS* 5.2.5, all *Connext DDS* libraries for Windows platforms (static release/debug, dynamic release/debug) now link with the dynamic Windows C Run-Time (CRT). Previously, the static *Connext DDS* libraries statically linked the CRT.

If you have an existing Windows project that was linking with the *Connext DDS* static libraries, you will need to change the RunTime Library settings:

- In Visual Studio, select C/C++, Code Generation, Runtime Library and use Multi-threaded DLL (**/MD**) instead of Multi-threaded (**/MT**) for static release libraries, and Multi-threaded Debug DLL

(**/MDd**) instead of Multi-threaded Debug (**/MTd**) for static debug libraries.

- For command-line compilation, use **/MD** instead of **/MT** for static release libraries, and **/MDd** instead of **/MTd** for static debug libraries.

In addition, you may need to ignore the static run-time libraries in their static configurations:

- In Visual Studio, select **Linker, Input** in the project properties and add **libcmtd;libcmt** to the **'Ignore Specific Default Libraries'** entry.
- For command-line linking, add **/NODEFAULTLIB:"libcmtd" /NODEFAULTLIB:"libcmt"** to the linker options.

9.3 Use Dynamic MFC Library, Not Static

To avoid communication problems in your *Connext DDS* application, use the dynamic MFC library, not the static version.

If you use the static version, your *Connext DDS* application may stop receiving DDS samples once the Windows sockets are initialized.

9.4 .NET API Requires Thread Affinity

To maintain proper concurrency control, .NET threads that call a *Connext DDS* API must correspond one-to-one with operating system threads. In most applications, this will always be the case. However, it may not be the case if the threads you are using are managed in a more advanced way.

If you intend to call *Connext DDS* APIs from explicitly managed threads, you must first call **Thread.BeginThreadAffinity()** in each such thread to ensure that it remains attached to a single operating system thread. See http://msdn.microsoft.com/en-us/library/system.threading.thread.beginthreadaffinity.aspx.

When you are done making RTI calls from a given thread, call **Thread.EndThreadAffinity()**.

In any case, be sure to consult the API Reference HTML documentation for more information about the thread safety contracts of the operations you use.

9.5 Support for Modern C++ API

Connext DDS provides two different C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs. The Modern C++ API is available for all Windows platforms.

Both C++03 and C++11 have been tested.

Notes:

- Support for C++03 is deprecated starting with release 6.1.0, which is the last release that supports non-C++11-compliant compilers. After release 6.1.0, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API is not affected and continues to support C++98 compilers (or newer).
- Only the default plugin is supported. The legacy plugin has been removed from *Code Generator* starting with release 6.1.0.

For more information, see Traditional vs. Modern C++, in the RTI Connext DDS Core Libraries User's Manual.

9.6 Multicast Support

Multicast is supported on all platforms and is configured out of the box. That is, the default value for the initial peers list (**NDDS_DISCOVERY_PEERS**) includes a multicast address. See the online documentation for more information.

9.7 Supported Transports

- **Shared memory:** Shared memory is supported and enabled by default. The Windows operating system manages the shared memory resources automatically. Cleanup is not required.
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported but disabled on architectures that use Visual Studio. The peers list (**NDDS_DISCOVERY_PEERS**) must be modified to support UDPv6. No Traffic Class support.
- **TCP/IPv4:** Supported on architectures that use Visual Studio. (This is *not* a built-in transport.)

9.8 Unsupported Features

These features are not supported on Windows platforms:

- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State is only supported on x64Win64VS2012 platforms
- Setting thread names by *Connext DDS* at the operating-system level in release mode

9.9 Monotonic Clock Support

The monotonic clock (described in *Clock Selection* in the RTI Connext DDS Core Libraries User's Manual) is supported on all Windows platforms.

9.10 Thread Configuration

See Table 9.6 Thread Settings for Windows Platforms, Table 9.7 Thread-Priority Definitions for Windows Platforms, and Table 9.8 Thread Kinds for Windows Platforms.

Table 9.6 Thread Settings for Windows Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread,	mask	OS default thread type
	priority	0
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	-3
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	-2
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO | DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	2
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 9.7 Thread-Priority Definitions for Windows Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	0
THREAD_PRIORITY_HIGH	3
THREAD_PRIORITY_ABOVE_NORMAL	2
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_BELOW_NORMAL	-2
THREAD_PRIORITY_LOW	-3

Table 9.8 Thread Kinds for Windows Platforms

Thread Kinds	Operating-System Configuration[a]
DDS_THREAD_SETTINGS_FLOATING_POINT	N/A
DDS_THREAD_SETTINGS_STDIO	
DDS_THREAD_SETTINGS_REALTIME_PRIORITY	
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE	

9.11 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on all Windows platforms in Table 9.1 Supported Windows Platforms.

For information on using this script, see *Building Applications Using CMake*, in the RTI Connext DDS Core Libraries Users Manual.

9.12 Durable Writer History and Durable Reader State Features

The Durable Writer History and Durable Reader State features have been tested with the following Windows architecture:

- x64Win64VS2012

To use the Durable Writer History and Durable Reader State features, you must install a relational database. Only MySQL® is supported.

[a]See Windows manuals for additional information.

For information on database setup and the versions supported, please see the RTI Connext DDS Core Libraries Database Setup.

9.13 Backtrace Support

To support the display of the backtrace on Windows systems, you need the **Dbghelp.dll** and **NtDll.dll** libraries. Without these libraries, the backtrace will not be available.

- To obtain the latest version of **DbgHelp.dll**, go to https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk and download Debugging Tools for Windows. Refer to "Calling the DbgHelp Library" for information on proper installation.
- **NtDll.dll** exports the Windows Native API. It is installed automatically during the installation of the Windows operating system.

When using release-mode libraries, backtrace support on Windows 32-bit architectures requires you to use the **/Oy-** optimization flag to disable "Frame-Pointer Omission" optimization.

See https://docs.microsoft.com/en-us/cpp/build/reference/oy-frame-pointer-omission?view=vs-2019.

See also Logging a Backtrace for Failures, in the RTI Connext DDS Core Libraries User's Manual.

9.14 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on these Windows architectures:

- i86Win32VS2015
- i86Win32VS2017
- x64Win64VS2015
- x64Win64VS2017

See *Remote Procedure Calls (RPC)* in the RTI Connext DDS Core Libraries User's Manual.

9.15 Libraries Required for Using Distributed Logger Support

RTI Distributed Logger is supported on all Windows platforms. Table 9.9 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need to use *Distributed Logger*.

Table 9.9 Additional Libraries for using RTI Distributed Logger

Language	Static[a]		Dynamic[b]	
	Release	Debug	Release	Debug
C	rtidlcz.lib	rtidlczd.lib	rtidlc.lib rtidlc.dll	rtidlcd.lib rtidlcd.dll
C++ (Traditional API)	rtidlcz.lib rtidlcppz.lib	rtidlczd.lib rtidlcppzd.lib	rtidlc.lib rtidlc.dll rtidlcpp.lib rtidlcpp.dll	rtidlcd.lib rtidlcd.dll rtidlcppd.lib rtidlcppd.dll
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

9.16 Libraries Required for Using Monitoring

To use the Monitoring APIs, reference the libraries in Table 9.10 Additional Libraries for Using Monitoring.

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext DDS* application is linked with the static release version of the *Connext DDS* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

If you are statically linking your application with DDS libraries and you want to add monitoring to your application, you will also need to statically link the monitoring library. The library cannot be loaded dynamically strictly through the QoS profile because it also depends on DDS to publish its data. Therefore, it depends on DDS; the DDS functionality would cause duplicate symbols to be found resulting, in the termination of the process.

Table 9.10 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[c]
Dynamic Release	rtimonitoring.lib rtimonitoring.dll
Dynamic Debug	rtimonitoringd.lib rtimonitoringd.dll

[a]These libraries are in <NDDSHOME>\lib\<architecture>.

[b]These libraries are in <NDDSHOME>\lib\<architecture>.

[c]These libraries are in <NDDSHOME>\lib\<architecture>.

Table 9.10 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Static Release	rtimonitoringz.lib Psapi.lib
Static Debug	rtimonitoringzd.lib Psapi.lib

9.17 Libraries Required for Using RTI Real-Time WAN Transport APIs

If you choose to use *RTI Real-Time WAN Transport*, you must download and install a separate package that contains the transport libraries. See the RTI Real-Time WAN Transport Installation Guide for details.

Using *Real-Time WAN Transport* requires one of the libraries in Table 9.11 Additional Libraries for Using RTI Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

For more information, see the "Enabling Real-Time WAN Transport" section in the *RTI Real-Time WAN Transport* part of the RTI Connext DDS Core Libraries User's Manual.

Table 9.11 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[b]
Dynamic Release	nddsrwt.dll
Dynamic Debug	nddsrwtd.dll
Static Release	nddsrwtz.lib
Static Debug	nddsrwtzd.lib

9.18 Libraries Required for Using RTI Secure WAN Transport APIs

To use the *RTI Secure WAN Transport* APIs, reference the libraries in Table 9.12 Additional Libraries for Using RTI Secure WAN Transport APIs. See the RTI Secure WAN Transport Release Notes and RTI Secure WAN Transport Installation Guide for details.

For details on the OpenSSL libraries, see 9.18.1 Location for OpenSSL Libraries on the next page.

[a]These libraries are in <NDDSHOME>\lib\<architecture>.

[b]These libraries are in <NDDSHOME>\lib\<architecture>.

Table 9.12 Additional Libraries for Using RTI Secure WAN Transport APIs

Library Format	RTI Secure WAN Transport Libraries[a]	OpenSSL Libraries	System Libraries
Dynamic Release	nddstransportwan.lib nddstransportwan.dll nddstransporttls.lib nddstransporttls.dll	libssl.lib libssl-<version>.dll libcrypto.lib libcrypto-<version>.dll	(none)
Dynamic Debug	nddstransporttlsd.lib nddstransporttlsd.dll nddstransportwand.lib nddstransportwand.dll		
Static Release	nddstransportwanz.lib nddstransporttlsz.lib	libsslz.lib libcryptoz.lib	crypt32.lib
Static Debug	nddstransportwanzd.lib nddstransporttlszd.lib		

9.18.1 Location for OpenSSL Libraries

OpenSSL **.lib** files are in <NDDSHOME>\third_party\openssl-1.1.1n\<architecture>\<format>\lib.

OpenSSL **.dll** files are in <NDDSHOME>\third_party\openssl-1.1.1n\<architecture>\<format>\bin.

Where:

- **<architecture>** is your architecture string, as listed in Table 9.1 Supported Windows Platforms on page 112, such as **x64Win64VS2017**.
- **<format>** is **debug**, **release**, **static_debug**, or **static_release**.

The **.dll** filenames have a <version> suffix. For example, **libssl-1_1-x64.dll** is for OpenSSL 1.1 on an x64 CPU.

9.19 Libraries Required for Using RTI TCP Transport APIs

To use the TCP Transport APIs, reference the libraries in Table 9.13 Additional Libraries for Using RTI TCP Transport APIs.

[a]These libraries are in <NDDSHOME>\lib\<architecture>.

Table 9.13 Additional Libraries for Using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries[a]
Dynamic Release	nddstransporttcp.lib nddstransporttcp.dll
Dynamic Debug	nddstransporttcpd.lib nddstransporttcpd.dll
Static Release	nddstransporttcpz.lib
Static Debug	nddstransporttcpzd.lib

If you are also using *RTI TLS Support*, see Table 9.14 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled. (Select the files appropriate for your chosen library format.)

For details on the OpenSSL libraries, see 9.18.1 Location for OpenSSL Libraries on the previous page.

Table 9.14 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[b]	OpenSSL Libraries	System Libraries
Dynamic Release	nddstls.lib nddstls.dll	libssl.lib libssl-<version>.dll libcrypto.lib libcrypto-<version>.dll	(none)
Dynamic Debug	nddstlsd.lib nddstlsd.dll		
Static Release	nddstlsz.lib	libsslz.lib libcryptoz.lib	crypt32.lib
Static Debug	nddstlszd.lib		

9.20 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, reference the libraries in Table 9.15 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 9.15 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries[c]
Dynamic Release	nddsmetp.lib nddsmetp.dll

[a]The libraries are in <NDDSHOME>\lib\<architecture>.

[b]The libraries are in <NDDSHOME>\lib\<architecture>.

[c]The libraries are in <NDDSHOME>\lib\<architecture>.

Table 9.15 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries[a]
Dynamic Debug	nddsmetpd.lib nddsmetpd.dll
Static Release	nddsmetpz.lib
Static Debug	nddsmetpzd.lib

9.21 Domain ID Support

On Windows platforms, you should avoid using ports 49152 through 65535 for inbound traffic. *Connext DDS*'s ephemeral ports (see *Ports Used for Discovery*, in the *Discovery* chapter of the RTI Connext DDS Core Libraries User's Manual) may be within that range (see https://msdn.microsoft.com/en-us/library/windows/desktop/ms737550(v=vs.85).aspx).

With the default RtpsWellKnownPorts settings, port 49152 corresponds to domain ID 167, so using domain IDs 168 through 232 on Windows platforms introduces the risk of a port collision and failure to create the *DomainParticipant* when using multicast discovery. You may see this error:

```
RTIOsapiSocket_bindWithIP:OS bind() failure, error 0X271D: An attempt was made to access a
socket in a way forbidden by its access permissions.
```

[a]The libraries are in <NDDSHOME>\lib\<architecture>.