RTI Connext Core Libraries

Platform Notes

Version 7.2.0

Trademarks

Copy and Use Restrictions

Third-Party Software

RTI software may contain independent, third-party software or code that are subject to third-party license terms and conditions, including open source license terms and conditions. Copies of applicable third-party licenses and notices are located at community.rti.com/documentation. IT IS YOUR RESPONSIBILITY TO ENSURE THAT YOUR USE OF THIRD-PARTY SOFTWARE COMPLIES WITH THE CORRESPONDING THIRD-PARTY LICENSE TERMS AND CONDITIONS.

Notices

Deprecations and Removals

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

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

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

Contents

Chapter 1 Introduction

This document provides platform-specific instructions that you will need to build and run *RTI® Connext®* applications.

For each supported OS, this document describes:

- Supported combinations of OS versions, CPUs, and compilers
- Building your application
  - Required *Connext* and system libraries
  - Required compiler and linker flags
  - Additional required libraries when using features such as Distributed Logger, Monitoring, Real-Time WAN Transport, TCP and TLS Support, and Zero Copy Transfer Over Shared Memory
  - Details on how the *Connext* libraries were built
- Running your application
- Whether or not certain features, APIs, and transports are supported, such as:
  - Modern C++, .NET, and Python APIs
  - Multicast
  - Transports
  - Monotonic clock
  - Durable Writer History and Durable Reader State
  - 'Find Package' CMake script
  - Backtraces
  - Remote Procedure Calls

- Thread configuration
- Other platform-specific information

To see all supported platforms, refer to the table of Supported Platforms tables in the RTI Connext Core Libraries Release Notes.

1.1 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

  This refers to the installation directory for *RTI® Connext®*. The default installation paths are:
  - macOS® systems:
    /Applications/rti_connext_dds-7.2.0
  - Linux systems, non-*root* user:
    /home/<*your user name*>/rti_connext_dds-7.2.0
  - Linux systems, *root* user:
    /opt/rti_connext_dds-7.2.0
  - Windows® systems, user without Administrator privileges:
    <*your home directory*>\rti_connext_dds-7.2.0
  - Windows systems, user with Administrator privileges:
    C:\Program Files\rti_connext_dds-7.2.0

  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-7.2.0\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/7.2.0/examples**
  - Linux systems: **/home/<*your user name*>/rti_workspace/7.2.0/examples**

- Windows systems: **<*your Windows documents folder*>\rti_workspace\7.2.0\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 Installation Guide*.

Chapter 2 Building Applications–Notes for All Platforms

This chapter provides general information on how to build *Connext* applications, for all platforms. Details such as exactly which libraries to link, compiler flags, etc., are in the platform-specific chapters in this document.

- First, make sure you've installed *Connext* 7.x.y. For installation instructions, see the RTI Connext Installation Guide.
- Make sure the **NDDSHOME** environment variable is set to the root directory of the *Connext* installation (such as **/home/user/rti_connext_dds-7.x.y** or **C:\Program Files\rti_connext_dds-7.x.y**). To confirm, type this at a command prompt:

  ```
  echo %NDDSHOME%
  ```

- To become familiar with *Connext* and the build process, follow the hands-on exercises in the RTI Connext Getting Started Guide.
- Review *this* chapter, which applies to all platforms.
- Build and test your applications on a Linux or Windows platform. They are both good starting points. See the instructions in either:
  - Chapter 3 Linux Platforms on page 10
  - Chapter 7 Windows Platforms on page 65
- Finally, build and run your applications on other platforms as needed. See the instructions in the other platform-specific chapters in this document.

To build a non-Java application using *Connext*, you must specify:

- NDDSHOME environment variable
- *Connext* header files

- *Connext* libraries to link
- Compatible system libraries
- Compiler options

To build Java applications using *Connext*, you must specify:

- NDDSHOME environment variable
- *Connext* JAR files
- Compatible Java virtual machine (JVM)
- Compiler options

2.1 Running on a Computer Not Connected to a Network

If you want to run two or more *Connext* applications on the same computer, *and* that computer is not connected to a network, you must set the environment variable NDDS_DISCOVERY_PEERS so that it will only use shared memory. For example:

```
set NDDS_DISCOVERY_PEERS=4@shmem://
```

(The number 4 is only an example. This is the maximum participant ID.)

2.2 Connext Header Files – All Platforms

You must include the appropriate *Connext* header files, As you will see in Table 2.1 Header Files to Include for Connext (All Platforms), the header files that need to be included depend on the API being used.

Table 2.1 Header Files to Include for Connext (All Platforms)

Connext API	Header Files
C	#include “ndds/ndds_c.h”
C++	#include “ndds/ndds_cpp.h”
C++/CLI, C#, Java	none

For the compiler to find the included files, the path to the appropriate include directories must be provided. Table 2.2 Include Paths for Compilation (All Platforms) lists the appropriate include path for use with the compiler. The exact path depends on where you installed *Connext*. See 1.1 Paths Mentioned in Documentation on page 3.

Table 2.2 Include Paths for Compilation (All Platforms)

Connext API	Include Path Directories
C and C++	<NDDSHOME>/include <NDDSHOME>/include/ndds
C++/CLI, C#, Java	none

You must also include the header files that define the data types you want to use in your application. For example, Table 2.3 Header Files to Include for User Data Types (All Platforms) lists the files to be include for type "Foo" (these are the filenames generated by *RTI Code Generator*, described in *Data Types and DDS Data Samples* chapter in the RTI Connext Core Libraries User's Manual).

Table 2.3 Header Files to Include for User Data Types (All Platforms)

Connext API	User Data Type Header Files
C and C++	#include "Foo.h" #include "FooSupport.h"
C++/CLI, C#, Java	none

2.3 Choosing the Right Libraries

2.3.1 Required Libraries

All required system and *Connext* libraries are listed in the chapters for each platform.

Choose between dynamic (shared) and static libraries. Do not mix the different types of libraries during linking. The benefit of linking against the dynamic libraries is that your final executables' sizes will be significantly smaller. You will also use less memory when you are running several *Connext* applications on the same node. However, shared libraries require more setup and maintenance during upgrades and installations.

To see if dynamic libraries are supported for your target platform, review the *Building Instructions* table in the chapter for that platform.

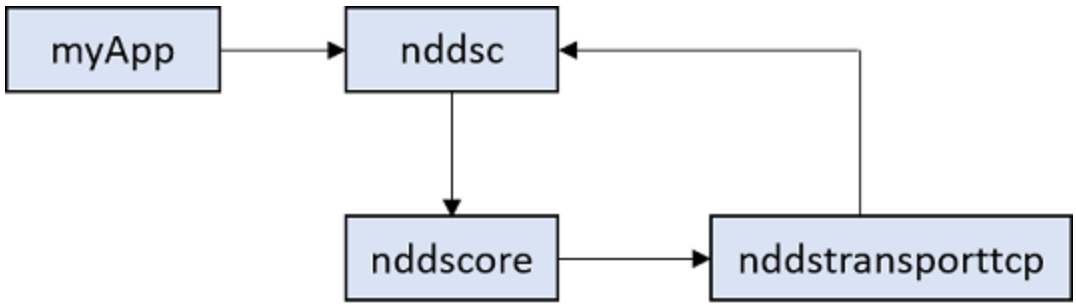
2.3.2 Mixing Static and Dynamic Libraries is not Supported

You must choose *either* static or dynamic linking. Mixing static and dynamic RTI libraries—for example, using RTI static core libraries and dynamic TCP Transport—is not supported.

The examples in this section are for Linux systems, but except for small differences in names, the same concepts apply to Windows and macOS systems.

Suppose you have a *Connext*-based application **myApp**, and you want to use the TCP Transport plugin. The library dependency looks something like Figure 2.1: Library Dependency below. This shows a simple and common situation, but make sure that the core libraries that your application uses are the same kinds of libraries that the TCP Transport plugin uses. For example, if **myApp** links statically with **nddsc**, but you load **nddstransporttcp** dynamically, there will be a mismatch between the libraries, potentially creating a dangerous situation. You must use static *or* dynamic linking, but not both.

Figure 2.1: Library Dependency



Important: Even if a combination of static and dynamic libraries seems to work, RTI cannot guarantee there won't be issues when running the *Connext* application.

2.4 Building for Java Platforms

Before building an application for a Windows or Linux Java platform, make sure that:

- *Connext* 7.x.y is installed (where 7.x.y stands for the version numbers of the current release).
- A supported JDK version is installed. See the *Supported Platforms* table at the beginning of the chapter for your platform.

Java Libraries: Certain Java archive (JAR) files must be on your classpath when running *Connext* applications.

Native Libraries: *Connext* for Java is implemented using Java Native Interface (JNI), so it is necessary to provide your *Connext* distributed applications with access to certain native shared libraries.

See the *Building Instructions* and *Running Instructions* tables in the chapter for your platform.

2.5 Building with CMake

Connext allows you to integrate the *Connext* libraries with build systems implemented using CMake®.

A “Find Package” CMake script is provided as part of the *Connext* installation. This script helps the build system find all the *RTI Connext* libraries and include directories needed by your application. So,

instead of setting the variables manually in your CMake scripts, you can call the *Connext* "Find Package CMake" script to set all the variables needed by your application.

Note: This script is not supported on all platforms. The chapter for your platform will show if it is supported.

You can find the script (**FindRTIConnextDDS.cmake**) in **<NDDSHOME>/resource/cmake**. To learn about the input and output variables, see the documentation included in the script.

Chapter 3 Linux Platforms

This release supports the Linux platforms in Table 3.1 Supported Linux Platforms in Connext 7.2.0 below.

Table 3.1 Supported Linux Platforms in Connext 7.2.0

Operating System	CPU	GLIBC	GLIBCXX	Compiler	RTI Architecture Abbreviation
CentOS 7.0	x64	2.17	6.0.19	gcc 4.8.2	x64Linux3gcc4.8.2
				Java Platform, Standard Edition AdoptOpenJDK 17.0.6	x64Linux3gcc4.8.2
Red Hat Enterprise Linux 7.0, 7.3, 7.5, 7.6	x64	2.17	6.0.19	gcc 4.8.2	x64Linux3gcc4.8.2
				Java Platform, Standard Edition AdoptOpenJDK 17.0.6	x64Linux3gcc4.8.2
Red Hat Enterprise Linux 8.0, 9.0	x64	2.28	6.0.25	gcc 7.3.0	x64Linux4gcc7.3.0
				Java Platform, Standard Edition AdoptOpenJDK 17.0.6	x64Linux4gcc7.3.0
Ubuntu 18.04, 20.04, 22.04 LTS	x64	2.27	6.0.25	gcc 7.3.0	x64Linux4gcc7.3.0
				Java Platform, Standard Edition AdoptOpenJDK 17.0.6	x64Linux4gcc7.3.0
Ubuntu 18.04 LTS	Arm v7	2.27	6.0.25	gcc 7.5.0	armv7Linux4gcc7.5.0
				Java Platform, Standard Edition AdoptOpenJDK 17.0.6	

Table 3.1 Supported Linux Platforms in Connext 7.2.0

Operating System	CPU	GLIBC	GLIBCXX	Compiler	RTI Architecture Abbreviation
Ubuntu 18.04 LTS, 22.04 LTS	Arm v8	2.27	6.0.25	gcc 7.3.0 Java Platform, Standard Edition AdoptOpenJDK 17.0.6	armv8Linux4gcc7.3.0

3.1 Building Applications for Linux Platforms

First, see the basic instructions in Chapter 2 Building Applications—Notes for All Platforms on page 5.

Then make sure that:

- *Connext* 7.x.y is installed (where 7.x.y stands for the version number of the current release). For installation instructions, refer to the RTI Connext Installation Guide.
- A "make" tool is installed. RTI recommends GNU Make. If you do not have it, you may be able to download it from your operating system vendor. Learn more at www.gnu.org/software/make/ or download from ftpmirror.gnu.org/make as source code.
- The **NDDSHOME** environment variable is set to the root directory of the *Connext* installation (such as **/home/user/rti_connext_dds-7.x.y**).
  - To confirm, type this at a command prompt:

```
echo $NDDSHOME
env | grep NDDSHOME
```

  - If it is not set or is set incorrectly, type:

```
export NDDSHOME=<correct directory>
```

3.1.1 Required Libraries and Compiler Flags

To compile a *Connext* application of any complexity, either modify the auto-generated makefile created by running *RTI Code Generator* or write your own makefile. See 3.1 Building Applications for Linux Platforms above for required compiler flags.

Table 3.2 Building Instructions for Linux Architectures lists the compiler flags and libraries you will need to link into your application.

Depending on which *Connext* features you want to use, you may need additional libraries; see 3.1.2 Additional Libraries for Other Features on page 13

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 3.2 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 64-bit architectures: -DRTI_LINUX -DRTI_UNIX -m64 For any Linux platform with GCC 6 or higher linker flag (see **Note** below table), also add: -no-pie 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 3.12 Backtrace Support on page 24)
	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 3.2 Building Instructions for Linux Architectures

API	Library Format	Required RTI Libraries or Jar Files[ab]	Required System Libraries	Required Compiler Flags
C	Static Release	libnddscorez.a libnddscz.a librticonnextmsgcz.a	-ldl -lm -lpthread -lrt	For 64-bit architectures: -DRTI_LINUX -DRTI_UNIX -m64 For any Linux platform with GCC 6 or higher linker flag (see **Note** below table), also add: -no-pie 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 3.12 Backtrace Support on page 24)
	Static Debug	libnddscorezd.a libnddsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnddscore.so libnddsc.so librticonnextmsgc.so		
	Dynamic Debug	libnddscored.so libnddscd.so librticonnextmsgcd.so		
Java	Release	nddsjava.jar rticonnextmsg.jar	N/A	None required
	Debug	nddsjavad.jar rticonnextmsgd.jar		

Note: For Linux platforms with GCC 6 or higher, it's possible to configure the compiler driver to link, by default, executables with PIE (position independent executable) support on amd64 and ppc64el architectures. Depending on the distributor of the GCC package, automatic PIE generation may or may not be enabled. To correctly generate backtraces, PIE executables cannot be used with RTI's libraries. This is due to Address Space Layout Randomization (ASLR), which prevents the correct generation of backtraces of our binaries on certain systems. For this reason, RTI has linked Linux executables using the **-no-pie** flag when the GCC version is 6 or higher. If you are using GCC 6 or higher, you must link the executable with **-no-pie** to prevent PIE generation and to correctly generate backtraces.

3.1.2 Additional Libraries for Other Features

3.1.2.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on all the platforms in Table 3.1 Supported Linux Platforms in Connext 7.2.0 on page 10.

To use the Distributed Logger APIs, link against the additional libraries in Table 3.3 Additional Libraries for using RTI Distributed Logger .

---

[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 3.3 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

3.1.2.2 Libraries Required for Monitoring

RTI Distributed Logger is supported on all the platforms in Table 3.1 Supported Linux Platforms in Connext 7.2.0 on page 10 . Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext* application is linked with the static release version of the *Connext* 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 3.4 Additional Libraries for Using Monitoring on the next page must appear *first* in the list of libraries to be linked.

Table 3.4 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

3.1.2.3 Libraries Required for Real-Time WAN Transport

If you choose to use *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 3.5 Additional Libraries for Using 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 Core Libraries User's Manual.

Table 3.5 Additional Libraries for Using 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

RTI Distributed Logger is supported on all the platforms in Table 3.1 Supported Linux Platforms in Connext 7.2.0 on page 10.

3.1.2.4 Libraries Required for TCP Transport and TLS Support

To use the TCP Transport APIs, link against the additional libraries in Table 3.6 Additional Libraries for using RTI TCP Transport APIs on the next page.

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

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

Table 3.6 Additional Libraries for using RTI TCP Transport APIs

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

If you are using *RTI TLS Support*, see Table 3.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled below. 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*®. If you choose to use *TLS Support*, it must be installed on top of a *Connext* 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 3.7 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	

3.1.2.5 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 3.8 Additional Libraries for Zero Copy Transfer Over Shared Memory on the next page.

[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-3.0.9/<architecture>/<format>/lib.

Table 3.8 Additional Libraries for Zero Copy Transfer Over Shared Memory

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

3.1.3 Linux Compatibility and Determining Factors

RTI has concluded that there are four factors that can be used to determine the compatibility of RTI's Linux core libraries on a specific Linux distribution or system. You can use this information to identify which *Connext* Linux libraries are suitable for your system. If a system matches the compatibility factors, RTI has a high level of confidence that the core libraries will work with no issues.

RTI has identified four Linux compatibility factors:

- CPU architecture (such as x64, Arm v8)
- Minimum GLIBC version
- GLIBCXX version
- Floating-Point scheme

3.1.3.1 Compatibility factors explained

The CPU architecture is the CPU family of the target system. Note that this important value is not for the *physical CPU* used to run, but the *configuration of the system where it will be executed*. For example, you may have an x64 CPU but your system kernel may run as if it were an x86 CPU. In this case, a 32-bit version of the *Connext* library should be selected.

The minimum GLIBC is the minimum required value of the GLIBC library used in the target system. If the target system's GLIBC version is less than the minimum version required by *Connext*, run-time errors can occur, such as undefined symbol errors.

The GLIBCXX range is the range of the Standard C++ Library that the target system must support. In some cases this value is a range and in others it’s a minimum value just like the minimum GLIBC support.

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

The floating-point scheme defines how the assembly code is generated relative to the floating-point registers and instructions; this should only be a concern on Arm v7 architectures. The options available are soft floating-point and hard floating-point. All newer architectures use hard floating-point.

Table 3.9 Compatibility Ranges

Library Name	CPU	Minimum GLIBC	GLIBCXX Range
x64Linux3gcc4.8.2	x64	2.17	6.0.15 <= X < 6.0.21
x64Linux4gcc7.3.0	x64	2.25	6.0.21 <= X
armv8Linux4gcc7.3.0	Arm v8	2.25	6.0.21 <= X

3.1.3.2 How to determine the GLIBC version on your target system

There are two ways to determine the GLIBC version in a target system. On most systems, you can run **ldd --version**. If the command **ldd** is not available, you must find where the **libc.so** library is located, then execute it. This will provide you the version of the library in the terminal. Note that you must perform this process on the target system in the case of cross-compiled architectures.

As an example, you can see the following output from an Ubuntu 20.04 system:

```
$ ldd --version
ldd (Ubuntu GLIBC 2.31-0ubuntu9.2) 2.31
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
$ ./lib/x86_64-linux-gnu/libc.so.6
GNU C Library (Ubuntu GLIBC 2.31-0ubuntu9.2) stable release version 2.31.
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Compiled by GNU CC version 9.3.0.
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
```

Given the output of both commands, we can say that the GLIBC version of this system is 2.31.

3.1.3.3 How to determine the GLIBCXX version on your target system

To determine the GLIBCXX version of the target system, you must find the **libstdc++.so.6.0.XX** library on your system. On some systems, you may have a **libstdc++.so** file, which is a symbolic link to the actual library.

The name of the libstdc++ library provides the version number, such as "**6.0.XX**" at the end of its name. Note that you must perform this process in the target system in the case of cross-compiled architectures. As an example, you can see the following output from an Ubuntu 20.04 system:

```
$ ls -l lib/x86_64-linux-gnu/libstdc++.so.6
lrwxrwxrwx 1 root root 19 May 29  2021 lib/x86_64-linux-gnu/libstdc++.so.6 ->
libstdc++.so.6.0.28
```

Given this output, we can determine that the GLIBCXX version for this system is 6.0.28.

3.1.4 How the Connext Libraries were Built

Table 3.10 Library-Creation Details for Linux Architectures provides details on how RTI built the Linux libraries. *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 3.10 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
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
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\"
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\"
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\"

Table 3.10 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
All supported Linux architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

3.2 Running Your Applications

For the environment variables that must be set at run time, see Table 3.11 Running Instructions for Linux Architectures below.

Table 3.11 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}

3.3 Support for the Modern C++ API

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

- The Modern C++ API requires C++11 compilers or newer.
- The Traditional C++ API supports C++98 compilers or newer.

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

The Modern C++ API is available for all supported Linux platforms.

3.4 Support for the .NET (C#) API

The C# API is supported on all Linux platforms. For more information on .NET, see the C# API Reference.

Note: The armv7Linux4gcc7.5.0 platform was tested by installing the native target library and explicitly setting LD_LIBRARY_PATH.

3.5 Support for the Python API

The Python API is only supported on one Linux platform: x64Linux4gcc7.3.0 (tested with Python 3.6-3.11). For more information, see the *Connext* Python API.

3.6 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.

3.7 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.

  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. This is *not* a built-in transport.

3.7.1 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* are in the range of 0x400000. For example:

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

The shared semaphore keys used by *Connext* 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*.

3.8 Monotonic Clock Support

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

3.9 Thread Configuration

Table 3.12 Thread Settings for Linux Platforms below lists the thread settings for Linux platforms.

See also: Table 3.13 Thread-Priority Definitions for Linux Platforms on the next page and Table 3.14 Thread Kinds for Linux Platforms on the next page.

3.9.1 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 platforms.

Table 3.12 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 platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	Empty CPU list
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION
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
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION

Table 3.12 Thread Settings for Linux 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	OS default thread stack size
	cpu_list	Empty CPU list
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION

Table 3.13 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 3.14 Thread Kinds for Linux Platforms

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

3.9.2 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

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

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.

3.10 Durable Writer History and Durable Reader State Features

The Durable Writer History and Durable Reader State features have been tested with all supported Linux architectures except armv8Linux4gcc7.3.0.

3.11 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on Linux platforms on *Intel* CPUs (see Table 3.1 Supported Linux Platforms in Connext 7.2.0 on page 10).

For information on using this script, see2.5 Building with CMake on page 8.

3.12 Backtrace Support

- If you are using GCC 6 or newer, you must link the executable with **-no-pie** in order to correctly generate backtraces. See the **Note** below Table 3.2 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**.

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

3.13 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature. It is only available for the C++11 API. It is supported all Linux architectures except x64Linux3gcc4.8.2.

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

Chapter 4 macOS Platforms

Table 4.1 Supported macOS Platforms in Connext 7.0.0 lists the architectures supported on macOS® operating systems.

Table 4.1 Supported macOS Platforms in Connext 7.0.0

Operating System	CPU	Compiler	RTI Architecture Abbreviation
macOS 11, 12 (host and target) [a]	x64	clang 12.0, 13.0	x64Darwin20clang12.0
		AdoptOpenJDK 17.0.6	
macOS 11 and 12 (target only) Requires Rosetta® 2 during installation. See 4.1 Installation Note for Arm v8 Platforms–Rosetta 2 Required below	Arm v8	clang 12.0, 13.0	arm64Darwin20clang12.0
		Java Platform, Standard Edition AdoptOpenJDK 17.0.6	

4.1 Installation Note for Arm v8 Platforms–Rosetta 2 Required

Rosetta® 2 must be installed and enabled before installing the host and target bundles for macOS 11 or 12 on an Arm v8 (M1) CPU.

Rosetta 2 is an Apple® tool for translating third party software applications; without it, you will see an error message when attempting to install *Connext*. 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*.

[a] Future releases may support a different version

4.2 Building Applications for macOS Platforms

Table 4.2 Building Instructions for macOS Architectures lists the compiler flags and libraries you will need to link into your application. Depending on which *Connext* features you want to use, you may need additional libraries; see 4.2.1 Additional Libraries for Other Features on the next page.

> 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 4.2 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* C/C++ libraries are in <NDDSHOME>/lib/<architecture>/.
<NDDSHOME> is where *Connext* is installed, see 1.1 Paths Mentioned in Documentation on page 3

Table 4.2 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		

4.2.1 Additional Libraries for Other Features

4.2.1.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on macOS platforms. Table 4.3 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

Table 4.3 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

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

[b]The *Connext* C/C++ libraries are in <NDDSHOME>/lib/<architecture>/.
<NDDSHOME> is where *Connext* is installed, see 1.1 Paths Mentioned in Documentation on page 3

Table 4.3 Additional Libraries for using RTI Distributed Logger

Language	Static		Dynamic	
	Release	Debug	Release	Debug
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

4.2.1.2 Libraries Required for Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext* application is linked with the static release version of the *Connext* 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 4.4 Additional Libraries for Using Monitoring must appear *first* in the list of libraries to be linked.

Table 4.4 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

4.2.1.3 Libraries Required for Real-Time WAN Transport

If you choose to use *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 4.5 Additional Libraries for Using Real-Time WAN Transport APIs. Select the file appropriate for your chosen library format.

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

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

Table 4.5 Additional Libraries for Using 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

4.2.1.4 Libraries Required for TCP Transport

To use the TCP Transport APIs, link against the additional libraries in Table 4.6 Additional Libraries for using RTI TCP Transport APIs. If you are using *RTI TLS Support*, see Table 4.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled. Select the files appropriate for your chosen library format.

Table 4.6 Additional Libraries for using RTI TCP Transport APIs

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

Table 4.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[c]	OpenSSL Libraries[d]
Dynamic Release	libnddstls.dylib	libssl.dylib libcrypto.dylib
Dynamic Debug	libnddstlsd.dylib	

[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]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-3.0.9/<architecture>/<format>/lib.

Table 4.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[a]	OpenSSL Libraries[b]
Static Release	libnddstlsz.a	libsslz.a libcryptoz.a
Static Debug	libnddstlszd.a	

4.2.1.5 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 4.8 Additional Libraries for Zero Copy Transfer Over Shared Memory .

Table 4.8 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

4.2.2 How the Connext Libraries were Built

Table 4.9 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 4.9 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

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

[b]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-3.0.9/<architecture>/<format>/lib.

Table 4.9 Library-Creation Details for macOS Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
x64Darwin20clang12.0	Release	-arch x86_64 -Wno-trigraphs -fpascal-strings -fasm-blocks -O -Wall -Wno-unknown-pragmas -DPtrIntType=long -DTARGET=\"x64Darwin20clang12.0\" -DNDEBUG
	Debug	-arch x86_64 -Wno-trigraphs -fpascal-strings -fasm-blocks -g -O -Wall -Wno-unknown-pragmas -DPtrIntType=long -DTARGET=\"x64Darwin20clang12.0\"
x64Darwin20clang12.0 for Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g

4.3 Running User Applications

Table 4.10 Running Instructions for macOS Architectures provides details on the environment variables that must be set at run time for a macOS architecture.

Table 4.10 Running Instructions for macOS Architectures

RTI Architecture	Library Format(Release & Debug)	Required Environment Variables[a]
arm64Darwin20clang12.0	Dynamic	DYLD_LIBRARY_PATH-H=${NDDSHOME}/lib/arm64Darwin20clang12.0:${DYLD_LIBRARY_PATH}
x64Darwin20clang12.0	Static	None required
	Dynamic	DYLD_LIBRARY_PATH-H=${NDDSHOME}/lib/x64Darwin20clang12.0:${DYLD_LIBRARY_ PATH}
x64Darwin20clang12.0 for Java	N/A	DYLD_LIBRARY_PATH-H=${NDDSHOME}/lib/x64Darwin20clang12.0:${DYLD_LIBRARY_ PATH}

4.4 Support for the Modern C++ API

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

[a]${NDDSHOME} is where *Connext* is installed. ${DYLD_LIBRARY_PATH} represents the value of the DYLD_LIBRARY_PATH variable prior to changing it to support *Connext*. 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).

- The Modern C++ API requires C++11 compilers or newer.
- The Traditional C++ API supports C++98 compilers or newer.

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

4.5 Support for the .NET (C#) API

The C# API is supported on macOS platforms with x64 and Arm v8 CPUs. For more information on .NET, see the C# API Reference.

4.6 Support for the Python API

The Python API is supported on platforms with Intel CPUs. For more information, see the *Connext* Python API.

4.7 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.

4.8 Transports

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

4.9 Unsupported Features

These features are not supported on macOS platforms:

- Controlling CPU Core Affinity
- Monotonic clock

4.10 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.

4.10.1 SIP and Java Applications

If you run *Connext* 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* 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* 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* 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* 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 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* 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 4.2.1.4 Libraries Required for TCP Transport on page 29.

4.10.2 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* 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 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>
```

4.11 Thread Configuration

See Table 4.11 Thread Settings for macOS Platforms and Table 4.12 Thread-Priority Definitions for macOS Platforms.

Table 4.11 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 4.12 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	

4.12 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on all macOS platforms with Intel CPUs.

For information on using this script, see 2.5 Building with CMake on page 8.

4.13 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 Core Libraries User's Manual.

4.14 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* 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
```

4.15 Support for Remote Procedure Calls (RPC)

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

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

Chapter 5 QNX Platforms

Table 5.1 Supported QNX Platforms for Connext 7.2.0 lists the architectures supported on QNX operating systems.[a]

Table 5.1 Supported QNX Platforms for Connext 7.2.0

Operating System	CPU	Compiler	RTI Architecture
QNX Neutrino 7.1	Arm v8 (64-bit)	qcc 8.3.0	armv8QNX7.1qcc_gpp8.3.0

5.1 Building Applications for QNX Platforms

The libraries on Arm CPUs require a hardware FPU in the processor and are compatible with systems that have hard-float libc. See Table 5.9 Library-Creation Details for QNX Architectures for compiler flag details.

Table 5.2 Building Instructions for QNX Architectures lists the libraries you will need to link into your application.

Depending on which *Connext* features you want to use, you may need additional libraries; see 5.1.2 Additional Libraries for Other Features on page 40.

Additional Documentation: You should also review the QNX chapter of the RTI Connext Core Libraries Getting Started Guide Addendum for Embedded Systems.

> 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.

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

Table 5.2 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 5.2 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		

5.1.1 Required Change for Building with C++ Libraries

The C++ libraries for QNX platforms 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 not necessary to use **-fno-rtti**, but doing so will not cause a problem.

5.1.2 Additional Libraries for Other Features

5.1.2.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on all QNX platforms.

Table 5.3 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

[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 5.3 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

5.1.2.2 Libraries Required for Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext* application is linked with the static release version of the *Connext* 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 5.4 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 5.4 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>.

5.1.2.3 Libraries Required for Real-Time WAN Transport

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 5.5 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 Core Libraries User's Manual.

Table 5.5 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

5.1.2.4 Libraries Required for TCP Transport APIs and TLS Support

To use the TCP Transport APIs, link against the additional libraries in Table 5.6 Additional Libraries for using RTI TCP Transport APIs .

Note: Not all platforms support the TCP Transport - see 5.5 Transports on page 45.

Table 5.6 Additional Libraries for using RTI TCP Transport APIs

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

If you are using *RTI TLS Support*, also see Table 5.7 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.

---

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

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

Table 5.7 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	

5.1.2.5 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 5.8 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 5.8 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

5.1.3 How the Connext Libraries were Built

Table 5.9 Library-Creation Details for QNX Architectures on the next page shows the compiler flags that RTI used to build the *Connext* libraries. This 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.

The details for building *user* applications are in 5.1 Building Applications for QNX Platforms on page 38.

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

[b]OpenSSL libraries are in <NDDSHOME>/third_party/openssl-3.0.9/<architecture>/<format>/lib.

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

Table 5.9 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv8QNX7.1qcc_gpp8.3.0	Static Release	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fPIC -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_gpp8.3.0\" -DNDEBUG -DRTI_QNX
	Static Debug	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fPIC -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_gpp8.3.0\" -DRTI_QNX
	Dynamic Release	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_gpp8.3.0\"-DNDEBUG -DRTI_QNX -fPIC
	Dynamic Debug	-Vgcc/8.3.0,gcc_ntoaarch64le -Y_gpp -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET=\"armv8QNX7.1qcc_gpp8.3.0\" -DRTI_QNX -fPIC

5.2 Running Your Application

Table 5.10 Running Instructions for QNX Architectures provides details on the environment variables that must be set at run time for a QNX architecture.

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

Table 5.10 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}[a]

5.3 Support for Modern C++ API

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

- The Modern C++ API requires C++11 compilers or newer.
- The Traditional C++ API supports C++98 compilers or newer.

[a] ${NDDSHOME} represents the root directory of your *Connext* installation. ${LD_LIBRARY_PATH} represents the value of the LD_LIBRARY_PATH variable prior to changing it to support *Connext*. 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.

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

5.4 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.

5.5 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* 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* 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* 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.

5.6 Unsupported Features

These features are not supported on QNX platforms:

- Java, .NET, and Python APIs
- Backtrace

- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State

5.7 Monotonic Clock Support

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

5.8 Thread Configuration

See Table 5.11 Thread Settings for QNX Platforms and Table 5.12 Thread-Priority Definitions for QNX Platforms.

Table 5.11 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
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	8
	stack_size	4 * 64 * 1024
	cpu_list	Empty CPU list (Supported on QNX platforms)
	cpu_rotation	DDS_THREAD_SETTINGS_CPU_NO_ROTATION

Table 5.11 Thread Settings for QNX Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
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 5.12 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

5.8.1 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.

5.9 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on QNX 7.1 platforms. For information on using this script, see2.5 Building with CMake on page 8.

5.10 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on all QNX platforms.

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

5.11 Restarting Applications on QNX Systems

Due to a limitation in the POSIX API, the allocation and the initialization of a shared memory mutex need to be done in separate steps.

The first (and only the first) *Connext* application that runs in the system using the shared-memory transport on a given domain will create a shared-memory mutex, in separate steps as described above, and subsequent *Connext* applications will attach to—but not create—this mutex, which is necessary to protect access to the shared-memory area across multiple processes.

It is possible under some extreme circumstances that the *Connext* application that creates the mutex crashes—or terminates ungracefully—having only partially created the mutex. If this occurs, other *Connext* applications will consider the mutex is still being created and will not be able to continue their execution, reporting a timeout error and indicating the mutex name.

If this situation occurs, you must manually delete the shared-memory mutex and its segment before relaunching any application in the same DDS domain. The files to delete are:

- **/dev/sem/RTIOsapiSharedMemoryMutex-<identifier>**
- **/dev/shmem/RTIOsapiSharedMemorySegment-<identifier>**

Chapter 6 VxWorks Platforms

Table 6.1 VxWorks Target Platforms for Connext 7.2.0 lists the architectures supported on VxWorks® operating systems. You can build a VxWorks application by cross-compiling from your development host.

Table 6.1 VxWorks Target Platforms for Connext 7.2.0

Operating System	CPU	Compiler	RTI Architecture[a]	
VxWorks 22.09	x64	llvm 13.0.1.3	For Kernel Modules:	x64Vx22.09llvm13.0.1.3
			For Real Time Processes:	x64Vx22.09llvm13.0.1.3_rtp

6.1 Building Applications for VxWorks Platforms

The following notes apply to VxWorks 7-based platforms, including VxWorks 22.09.

- Compiling a *Connext* application for VxWorks depends on the development platform. For more information, such as specific compiler flags, see the *VxWorks Programmer's Guide*. Table 6.7 Library-Creation Details for VxWorks Architectures on page 56 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*.

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

- Required Makefile Change

  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* libraries for VxWorks 7.0 for x64 CPUs.

- 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* 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* 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* libraries, and another static DKM containing the subscriber application + *Connext* libraries. When those two modules are loaded into the kernel, all the *Connext* 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* 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.

6.1.1 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* libraries (in particular, libnddsc) export a number of symbols that exceed this upper bound.

Dynamic libraries *are* available for VxWorks systems with RTP mode.

6.1.2 Required Libraries and Compiler Flags

First, see the basic instructions in Chapter 2 Building Applications—Notes for All Platforms on page 5.

Table 6.2 Building Instructions for VxWorks Architectures on the next page lists the libraries you will need to link into your application and the required compiler flags.

Depending on which *Connext* features you want to use, you may need additional libraries; see 6.1.3 Additional Libraries for Other Features on page 53.

Additional Documentation: See the RTI Connext Core Libraries Getting Started Guide Addendum for Embedded Systems.

> 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 6.2 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 -DRTI_CLANG -DRT_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* C/C++ libraries are in <NDDSHOME>/lib/<*architecture*>.

Table 6.2 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 -DRTI_CLANG -DRT_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		

6.1.3 Additional Libraries for Other Features

6.1.3.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported all VxWorks architectures. Table 6.3 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need in order to use *Distributed Logger*.

Table 6.3 Additional Libraries for using RTI Distributed Logger

Language	Static[c]		Dynamic[d]	
	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

6.1.3.2 Libraries Required for Monitoring

Make sure you are consistent in your use of static, dynamic, debug and release versions of the libraries. For example, if your *Connext* application is linked with the static release version of the *Connext*

---

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

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

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

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

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 6.4 Additional Libraries for Monitoring must appear *first* in the list of libraries to be linked.

Table 6.4 Additional Libraries for Monitoring

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

6.1.3.3 Libraries Required for Real-Time WAN Transport APIs

If you choose to use *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 6.5 Additional Libraries for Using 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 Core Libraries User's Manual.

---

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

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

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

Table 6.5 Additional Libraries for Using 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

6.1.3.4 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 6.6 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 6.6 Additional Libraries for Zero Copy Transfer Over Shared Memory

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

6.1.4 How the Connext Libraries were Built

Table 6.2 Building Instructions for VxWorks Architectures on page 52 shows the compiler flags that RTI used to build the Connext libraries. This 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.

The details for building *user* applications are in 6.1 Building Applications for VxWorks Platforms on page 49.

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

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

Table 6.7 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Vx22.09llvm13.0.1.3	Static Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET=\"x64Vx22.09llvm13.0.1.3\" -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 -D__ELF__ -D__VXWORKS__ -D__vxworks --target=x86_64-wrs-vxworks -m64 -march=core2 -mcmodel=kernel -mno-implicit-float -mno-red-zone -fno-builtin -fno-omit-frame-pointer -fno-strict-aliasing -ftls-model=local-exec -nostdlibinc -nostdinc++ -Walll -Wno-unknown-pragmas -Werror=implicit-function-declaration -O -DNDEBUG -std=c11
	Static Debug	-DCPU=_VX_CORE -DPtrIntType=long -DTARGET=\"x64Vx22.09llvm13.0.1.3\" -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 -D__ELF__ -D__VXWORKS__ -D__vxworks -O0 --target=x86_64-wrs-vxworks -m64 -march=core2 -mcmodel=kernel -mno-implicit-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=c11
	Dynamic Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET=\"x64Vx22.09llvm13.0.1.3\" -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__vxworks -O --target=x86_64-wrs-vxworks -m64 -march=core2 -mcmodel=kernel -mno-implicit-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=\"x64Vx22.09llvm13.0.1.3\" -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__vxworks -O0 --target=x86_64-wrs-vxworks -m64 -march=core2 -mcmodel=kernel -mno-implicit-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=c11

Table 6.7 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Vx22.09llvm13.0.1.3_rtp	Static Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET=\"x64Vx22.09llvm13.0.1.3_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-vxworks -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=\"x64Vx22.09llvm13.0.1.3_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-vxworks -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=c11
	Dynamic Release	-DCPU=VX_CORE -DPtrIntType=long -DTARGET=\"x64Vx22.09llvm13.0.1.3_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-vxworks -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=\"x64Vx22.09llvm13.0.1.3_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-vxworks -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

6.2 Running User Applications

Table 6.8 Running Instructions for VxWorks Architectures below provides details on the environment variables that must be set at runtime for a VxWorks architecture.

Table 6.8 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>" [a]
	Static	None required

6.3 Known Defects

There are no known defects in VxWorks that would affect the supported architectures.

6.4 Increasing the Stack Size

Connext 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.

6.5 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.

[a] 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.

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

The *Connext* Professional, Research, and LM packages include support for the Request-Reply Communication Pattern, for all platforms in Table 6.1 VxWorks Target Platforms for Connext 7.2.0 on page 49 and all programming languages.

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* 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* 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* will be loaded once per DKM.
- Perform a partial linkage: This involves building your application without linking against the *Connext* libraries. Later, at load time, you will need to load into the kernel the required *Connext* libraries and your application DKM. This is recommended if you plan to have more than one DKM using *Connext*.

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

6.7 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* applications running on other systems that were talking to the restarted application. If all the *Connext* applications are restarted, there should be no problem.

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

All *Connext* 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 increment 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.

6.8 Support for Modern C++ API

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

- The Modern C++ API requires C++11 compilers or newer.
- The Traditional C++ API supports C++98 compilers or newer.

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

6.9 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.

6.10 Transports

- **Shared memory:** Shared memory is supported and enabled by default on all VxWorks architectures. See also:
  - 6.10.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID below
  - 6.10.2 How To Run Connext DDS Libraries in Kernels Built without Shared Memory on the next page
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported. No Traffic Class support.
- **TCP/IPv4:** Not supported.

6.10.1 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.

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

Since *Connext* 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* 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.

6.11 Unsupported Features

These features are not supported on any VxWorks platforms:

- Java, .NET, and Python APIs
- Backtrace
- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script

- Remote Procedure Calls (RPCs)
- TCP v4 transport

6.12 Monotonic Clock Support

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

6.13 Use of Real-Time Clock

Starting with 5.3.0, *Connext* 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.

6.14 Thread Configuration

See these tables:

- Table 6.9 Thread Setting for VxWorks Platforms below
- Table 6.10 Thread-Priority Definitions for VxWorks Platforms on the next page
- Table 6.11 Thread Kinds for VxWorks Platforms on page 64

Table 6.9 Thread Setting for VxWorks Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting for kernel tasks and RTP threads
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

Table 6.9 Thread Setting for VxWorks Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting for kernel tasks and RTP threads
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 6.10 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 6.11 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

[a]See VxWorks manuals for more information.

Chapter 7 Windows Platforms

This release supports the Windows platforms in Table 7.1 Supported Windows Platforms in Connext 7.2.0.

Table 7.1 Supported Windows Platforms in Connext 7.2.0

Operating System	CPU	Visual Studio® Version	RTI Architecture Abbreviation	.NET Version	JDK Version
Windows 10, 11	x64	VS 2015 Update 3	x64Win64VS2015	.NET Standard 2.0	AdoptOpenJDK 17.0.6
		VS 2017 Update 2 VS 2019 Version 16.0.0 VS 2022	x64Win64VS2017		
Windows Server 2012 R2		VS 2015	x64Win64VS2015		
Windows Server 2016		VS 2015 Update 3	x64Win64VS2015		
		VS 2017 Update 2 VS 2019 Version 16.0.0 VS 2022	x64Win64VS2017		

Note regarding C# API Support: The C# API is supported on Windows 10 systems, but it doesn't support Visual Studio 2015 for development. Development is supported on Visual Studio 2017 or newer, Visual Studio Code, and the .NET command-line interface. For more information on .NET, see the C# API Reference.

7.1 Building Applications for Windows Platforms

First, see the basic instructions in Chapter 2 Building Applications—Notes for All Platforms on page 5.

Then make sure that:

- Supported versions of Windows and Visual Studio are installed (see Table 7.1 Supported Windows Platforms in Connext 7.2.0 on the previous page).
- You are using the *dynamic* MFC Library (not static).

  To avoid communication problems in your *Connext* application, use the dynamic MFC library, not the static version. (If you use the static version, your *Connext* application may stop receiving DDS samples once the Windows sockets are initialized.)

To compile a *Connext* application of any complexity, use a project file in Microsoft Visual Studio. The project settings are described below.

7.1.1 Using Visual Studio

1. Select the multi-threaded project setting:
   a. From the **Project** menu, select **Properties**.
   b. Select the **C/C++** folder.
   c. Select **Code Generation**.
   d. Set the **Runtime Library** field to one of the options in Table 7.2 Runtime Library Settings for Visual Studio below.

Table 7.2 Runtime Library Settings for Visual Studio

If you are using this Library Format...	Set the Runtime Library field to...
Release version of static libraries	Multi-threaded DLL (/MD)
Debug version of static libraries	Multi-threaded Debug DLL (/MDd)
Release version of dynamic libraries	Multi-threaded DLL (/MD)
Debug version of dynamic libraries	Multi-threaded Debug DLL (/MDd)

2. Link against the *Connext* libraries:
   a. Select the **Linker** folder in the Project, Properties dialog box.
   b. Select the **Input** properties.
   c. See which libraries you need by consulting these tables:

- Table 7.3 Building Instructions for Windows Host Architectures
- Table 7.4 Building Instructions for Windows Target Architectures

Choose whether to link with *Connext*'s static or dynamic libraries, and whether you want debugging symbols on or off.

Add the libraries to the *beginning* of the Additional Dependencies field. Be sure to use a *space* as a delimiter between libraries, *not* a comma.

Depending on which *Connext* features you want to use, you may need additional libraries; see 7.1.4 Additional Libraries for Other Features on page 72

d. Select the **General** properties.

e. Add the following to the **Additional library path** field (replace <architecture> to match your installed system):

```
$(NDDSHOME)\lib\<architecture>
```

3. Specify the path to the *Connext* header files:

   a. Select the **C/C++** folder.

   b. Select the **General** properties.

   c. In the **Additional include directories:** field, add paths to the "include" and "include\ndds" directories. For example: (your paths may differ, depending on where you installed *Connext*).

```
c:\Program Files\rti_connext_dds-7.x.y\include\
c:\Program Files\rti_connext_dds-7.x.y\include\ndds
```

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 7.3 Building Instructions for Windows Host Architectures

API	Library Format	RTI Libraries or Jar Files [ab]	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 /D "WIN32_LEAN_AND_MEAN
	Static Debug	nddscorezd.lib nddsczd.lib rticonnextmsgczd.lib		/D “RTI_WIN32” /MDd /D "WIN32_LEAN_AND_MEAN
	Dynamic Release	nddscore.lib nddsc.lib rticonnextmsgc.lib		/D “RTI_WIN32” /D “NDDS_DLL_VARIABLE” /MD /D "WIN32_LEAN_AND_MEAN
	Dynamic Debug	nddscored.lib nddscd.lib rticonnextmsgcd.lib		/D “RTI_WIN32” /D “NDDS_DLL_VARIABLE” /MDd /D "WIN32_LEAN_AND_MEAN

[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.

Table 7.3 Building Instructions for Windows Host Architectures

API	Library Format	RTI Libraries or Jar Files [ab]	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 /D "WIN32_LEAN_AND_MEAN
	Static Debug	nddscorezd.lib nddsczd.lib nddscppzd.lib or nddscpp2zd.lib rticonnextmsgcppzd.lib or rticonnextmsgcpp2zd.lib		/D “RTI_WIN32” /MDd /D "WIN32_LEAN_AND_MEAN
	Dynamic Release	nddscore.lib nddsc.lib nddscpp.lib or nddscpp2.lib rticonnextmsgcpp.lib or rticonnextmsgcpp2.lib		/D “RTI_WIN32” /D “NDDS_DLL_VARIABLE” /MD /D "WIN32_LEAN_AND_MEAN
	Dynamic Debug	nddscored.lib nddscd.lib nddscppd.lib or nddscpp2d.lib rticonnextmsgcppd.lib or rticonnextmsgcpp2d.lib		/D “RTI_WIN32” /D “NDDS_DLL_VARIABLE” /MDd /D "WIN32_LEAN_AND_MEAN
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.

Table 7.4 Building Instructions for Windows Target Architectures

API	Library Format	RTI Libraries or Jar Files [a]	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” /D “WIN32_LEAN_AND_MEAN”
	Static Debug	nddscorezd.lib nddsczd.lib rticonnextmsgczd.lib		/Gd /MDd /D “WIN32” /D “RTI_WIN32” /D “WIN32_LEAN_AND_MEAN”
	Dynamic Release	nddscore.lib nddsc.lib rticonnextmsgc.lib		/Gd /MD /D “WIN32” /D “NDDS_DLL_VARIABLE” /D “RTI_WIN32” /D “NDEBUG” /D “WIN32_LEAN_AND_MEAN”
	Dynamic Debug	nddscored.lib nddscd.lib rticonnextmsgcd.lib		/Gd /MDd /D “WIN32” /D “NDDS_DLL_VARIABLE” /D “RTI_WIN32” /D “WIN32_LEAN_AND_MEAN”
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” /D “WIN32_LEAN_AND_MEAN”
	Static Debug	nddscorezd.lib nddsczd.lib nddscppzd.lib or nddscpp2zd.lib rticonnextmsgcppzd.lib or rticonnextmsgcpp2zd.lib		/Gd /EHsc /MDd /D “WIN32” /D “RTI_WIN32” /D “WIN32_LEAN_AND_MEAN”

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

Table 7.4 Building Instructions for Windows Target Architectures

API	Library Format	RTI Libraries or Jar Files [a]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs) (cont'd)	Dynamic Release	nddscore.lib nddsc.lib nddscpp.lib or nddscpp2.lib rticonnextmsgcpp.lib or rticonnextmsgcpp2.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	/Gd /EHsc /MD /D “WIN32” /D “NDDS_DLL_VARIABLE” /D “RTI_WIN32” /D “NDEBUG” /D “WIN32_LEAN_AND_MEAN”
	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” /D “WIN32_LEAN_AND_MEAN”
Java	Release	nddsjava.jar rticonnextmsg.jar	N/A	N/A
	Debug	nddsjavad.jar rticonnextmsgd.jar		

7.1.2 Linking Windows C Run-Time Libraries

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

If you have an existing Windows project that was linking with the *Connext* 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:

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

- 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.

7.1.3 Use the Dynamic MFC Library, Not Static

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

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

7.1.4 Additional Libraries for Other Features

7.1.4.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on all Windows platforms. Table 7.5 Additional Libraries for using RTI Distributed Logger lists the additional libraries you will need to use *Distributed Logger*.

Table 7.5 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

7.1.4.2 Libraries Required for Monitoring

To use the Monitoring APIs, reference the libraries in Table 7.6 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* application is linked with the static release version of the *Connext*

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

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

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 7.6 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries[a]
Dynamic Release	rtimonitoring.lib rtimonitoring.dll
Dynamic Debug	rtimonitoringd.lib rtimonitoringd.dll
Static Release	rtimonitoringz.lib Psapi.lib
Static Debug	rtimonitoringzd.lib Psapi.lib

7.1.4.3 Libraries Required for Real-Time WAN Transport

If you choose to use *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 7.1.4 Additional Libraries for Other Features. 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 Core Libraries User's Manual.

Table 7.7 Additional Libraries for Using Real-Time WAN Transport APIs

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

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

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

Table 7.7 Additional Libraries for Using Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries[a]
Static Release	nddsrwtz.lib
Static Debug	nddsrwtzd.lib

For details on the OpenSSL libraries, see 7.1.6 Location of OpenSSL Libraries on page 76.

7.1.4.4 Libraries Required for RTI TCP Transport

To use the TCP Transport APIs, reference the libraries in Table 7.8 Additional Libraries for Using RTI TCP Transport APIs.

Table 7.8 Additional Libraries for Using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries[b]
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 7.9 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled. (Select the files appropriate for your chosen library format.)

Table 7.9 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[c]	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		

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

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

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

Table 7.9 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries[a]	OpenSSL Libraries	System Libraries
Static Release	nddstlsz.lib	libsslz.lib libcryptoz.lib	crypt32.lib
Static Debug	nddstlszd.lib		

For details on the OpenSSL libraries, see 7.1.6 Location of OpenSSL Libraries on the next page.

7.1.4.5 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, reference the libraries in Table 7.10 Additional Libraries for Zero Copy Transfer Over Shared Memory.

Table 7.10 Additional Libraries for Zero Copy Transfer Over Shared Memory

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

7.1.5 How the Connext Libraries were Built

Table 7.11 Library-Creation Details for Windows Architectures shows the compiler flags that RTI used to build the Connext libraries. This 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.

The details for building *user* applications are in 7.1 Building Applications for Windows Platforms on page 66

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

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

Table 7.11 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
x64 Windows architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

7.1.6 Location of OpenSSL Libraries

The OpenSSL libraries are installed here:

- OpenSSL **.lib** files are in <NDDSHOME>\third_party\openssl-3.0.9\<architecture>\<format>\lib.
- OpenSSL **.dll** files are in <NDDSHOME>\third_party\openssl-3.0.9\<architecture>\<format>\bin.

Where:

- <architecture> is your architecture string, as listed in Table 7.1 Supported Windows Platforms in Connext 7.2.0 on page 65, 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.

7.2 Running Your Applications

For the environment variables that must be set at run time, see Table 7.12 Running Instructions for Windows Architectures below.

Table 7.12 Running Instructions for Windows Architectures

RTI Architecture	Library Format	Environment Variables[a]
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%

7.2.1 Requirements when Using 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 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.

[a] **%Path%** represents the value of the **Path** variable prior to changing it to support *Connext*. 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.

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 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 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 — 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 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, Frameworks, and Redistributables" for Microsoft Visual C++ Redistributable for Visual Studio 2022".

7.3 Support for the Modern C++ API

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

- The Modern C++ API requires C++11 compilers or newer.
- The Traditional C++ API supports C++98 compilers or newer.

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

7.4 Support for the .NET (C#) API

The C# API is supported on all Windows platforms. For more information on .NET, see the C# API Reference.

For x64Win64VS2015: The .NET API is supported on Windows 10, but it doesn't support Visual Studio 2015 for development. Development is supported in Visual Studio 2017 or newer, Visual Studio Code, and the .NET CLI.

7.5 Support for the Python API

The Python API is supported on all Windows platforms (tested with Python 3.6-3.11). For more information, see the *Connext* Python API.

7.6 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.

7.7 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.)

7.8 Unsupported Features

These features are not supported on Windows platforms:

- Controlling CPU Core Affinity
- Setting thread names by *Connext* at the operating-system level in release mode

7.9 Monotonic Clock Support

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

7.10 Thread Configuration

See these tables:

- Table 7.13 Thread Settings for Windows Platforms
- Table 7.14 Thread-Priority Definitions for Windows Platforms
- Table 7.15 Thread Kinds for Windows Platforms

Table 7.13 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 7.14 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

Table 7.14 Thread-Priority Definitions for Windows Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_BELOW_NORMAL	-2
THREAD_PRIORITY_LOW	-3

Table 7.15 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	

7.11 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on all Windows platforms.

For information on using this script, see 2.5 Building with CMake on page 8.

7.12 Durable Writer History and Durable Reader State Features

The Durable Writer History and Durable Reader State features have been tested with all supported Windows platforms.

7.13 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 get 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.

[a]See Windows manuals for additional information.

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 Core Libraries User's Manual.

7.14 Support for Remote Procedure Calls (RPC)

RPC is an experimental feature available only for the C++11 API. It is supported on all Windows platforms.

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

7.15 Domain ID Support

On Windows platforms, you should avoid using ports 49152 through 65535 for inbound traffic. *Connext*'s ephemeral ports (see *Ports Used for Communication*, in the *Getting Applications to Discover Each Other* part of the RTI Connext 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.
```