

RTI Connex Core Libraries

Platform Notes

Version 7.3.0



Trademarks

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

Copy and Use Restrictions

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

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

1.1 Paths Mentioned in Documentation	3
--	---

Chapter 2 Building Applications—Notes for All Platforms

2.1 Running on a Computer Not Connected to a Network	6
2.2 Connex Header Files — All Platforms	6
2.3 Choosing the Right Libraries	7
2.3.1 Required Libraries	7
2.3.2 Mixing Static and Dynamic Libraries is not Supported	7
2.4 Building for Java Platforms	8
2.5 Building with CMake	8

Chapter 3 AIX Platforms

3.1 Building Applications for AIX Platforms	10
3.1.1 Additional Libraries for Other Features	12
3.1.2 How the Connex Libraries were Built	13
3.2 Running User Applications	14
3.3 Multicast Support	14
3.4 Transports	14
3.4.1 Notes for Using Shared Memory	14
3.5 Unsupported Features	15
3.6 Thread Configuration	15
3.6.1 Changing Thread Priority	17

Chapter 4 Android Platforms

4.1 Building Applications for Android Platforms	18
4.1.1 Required Libraries and Compiler Flags	18
4.1.2 Additional Libraries for Other Features	20
4.1.3 Target Configuration	22

4.1.4 ‘Release’ and ‘Debug’ Terminology	22
4.1.5 How the Connex Libraries were Built	22
4.2 Running Your Applications	23
4.3 Support for Modern C++ API	24
4.4 Multicast Support	24
4.5 Transports	24
4.6 Unsupported Features	24
4.7 Monotonic Clock Support	25
4.8 Thread Configuration	25
4.9 Support for Remote Procedure Calls (RPCs)	26
4.10 Third-Party Software Versions used for Android 12 Development and Testing	27
Chapter 5 Linux Platforms	
5.1 Building Applications for Linux Platforms	30
5.1.1 Required Libraries and Compiler Flags	30
5.1.2 Additional Libraries for Other Features	33
5.1.3 Linux Compatibility and Determining Factors	36
5.1.4 How the Connex Libraries were Built	38
5.2 Running Your Applications	41
5.3 Support for the Modern C++ API	41
5.4 Support for the .NET (C#) API	41
5.5 Support for the Python API	41
5.6 Multicast Support	42
5.7 Transports	42
5.7.1 Shared Memory Support	42
5.8 Limitations of FACE Architectures	42
5.9 Monotonic Clock Support	43
5.10 Thread Configuration	43
5.10.1 Support for Controlling CPU Core Affinity for RTI Threads	44
5.10.2 Using REALTIME_PRIORITY	45
5.11 Durable Writer History and Durable Reader State Features	46
5.12 Support for 'Find Package' CMake Script	46
5.13 Backtrace Support	46
5.14 Support for Remote Procedure Calls (RPC)	47
Chapter 6 macOS Platforms	
6.1 Building Applications for macOS Platforms	48
6.1.1 Additional Libraries for Other Features	51

6.1.2 How the Connex Libraries were Built	54
6.2 Running User Applications	55
6.3 Support for the Modern C++ API	55
6.4 Support for the .NET (C#) API	55
6.5 Support for the Python API	56
6.6 Multicast Support	56
6.7 Transports	56
6.8 Unsupported Features	56
6.9 System Integrity Protection (SIP)	56
6.9.1 SIP and Java Applications	56
6.9.2 SIP and Connex Tools, Infrastructure Services, and Utilities	58
6.10 Thread Configuration	58
6.11 Support for 'Find Package' CMake Script	59
6.12 Backtrace Support	59
6.13 Resolving NDDUtility_sleep() Issues	60
6.14 Support for Remote Procedure Calls (RPC)	60
Chapter 7 QNX Platforms	
7.1 Building Applications for QNX Platforms	62
7.1.1 Required Change for Building with C++ Libraries	64
7.1.2 Additional Libraries for Other Features	64
7.1.3 How the Connex Libraries were Built	67
7.2 Running Your Application	70
7.3 Support for Modern C++ API	70
7.4 Multicast Support	71
7.5 Transports	71
7.6 Unsupported Features	71
7.7 Monotonic Clock Support	72
7.8 Thread Configuration	72
7.8.1 Support for Controlling CPU Core Affinity for RTI Threads	73
7.9 Support for 'Find Package' CMake Script	73
7.10 Support for Remote Procedure Calls (RPC)	73
7.11 Restarting Applications on QNX Systems	74
Chapter 8 VxWorks Platforms	
8.1 Building Applications for VxWorks Platforms	75
8.1.1 Libraries for RTP Mode on VxWorks Systems	76
8.1.2 Required Libraries and Compiler Flags	77

8.1.3 Additional Libraries for Other Features	79
8.1.4 How the ConnexT Libraries were Built	81
8.2 Running User Applications	84
8.3 Known Defects	84
8.4 Increasing the Stack Size	84
8.5 Enabling Floating Point Coprocessor in Kernel Tasks	84
8.6 Downloadable Kernel Modules (DKM) for Kernel Mode on VxWorks Systems	85
8.7 Requirement for Restarting Applications	85
8.8 Support for Modern C++ API and Remote Procedure Calls (RPCs)	86
8.9 Multicast Support	86
8.10 Transports	86
8.10.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID	87
8.10.2 How To Run ConnexT Libraries in Kernels Built without Shared Memory	87
8.11 Unsupported Features	88
8.12 Monotonic Clock Support	88
8.13 Use of Real-Time Clock	88
8.14 Thread Configuration	88
Chapter 9 Windows Platforms	
9.1 Building Applications for Windows Platforms	92
9.2 Configuring the Build of Your ConnexT Application	92
9.2.1 Additional Libraries for Other Features	95
9.2.2 How the ConnexT Libraries were Built	98
9.2.3 Location of OpenSSL Libraries	100
9.3 Running Your Applications	100
9.4 Support for the Modern C++ API	101
9.5 Support for the .NET (C#) API	102
9.6 Support for the Python API	102
9.7 Multicast Support	102
9.8 Transports	102
9.9 Unsupported Features	102
9.10 Monotonic Clock Support	103
9.11 Thread Configuration	103
9.12 Support for 'Find Package' CMake Script	104
9.13 Durable Writer History and Durable Reader State Features	105
9.14 Backtrace Support	105
9.15 Support for Remote Procedure Calls (RPC)	105

9.16 Domain ID Support	105
------------------------------	-----

Chapter 1 Introduction

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

For each supported OS, this document describes:

- Supported combinations of OS versions, CPUs, and compilers
- Building your application
 - Required *Connex* 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 *Connex* 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 Connex Core Libraries Release Notes](#). These tables show which RTI products are supported for each platform.

1.1 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

This refers to the installation directory for *RTI® Connexxt®*. The default installation paths are:

- macOS® systems:
/Applications/rti_connexxt_dds-7.3.0
- Linux systems, non-*root* user:
/home/<your user name>/rti_connexxt_dds-7.3.0
- Linux systems, *root* user:
/opt/rti_connexxt_dds-7.3.0
- Windows® systems, user without Administrator privileges:
<your home directory>\rti_connexxt_dds-7.3.0
- Windows systems, user with Administrator privileges:
C:\Program Files\rti_connexxt_dds-7.3.0

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_connexxt_dds-7.3.0\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/7.3.0/examples
- Linux systems: /home/<your user name>/rti_workspace/7.3.0/examples

- Windows systems: **<your Windows documents folder>\rti_workspace\7.3.0\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 Connex Installation Guide*.

Chapter 2 Building Applications—Notes for All Platforms

This chapter provides general information on how to build *Connex*t 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 *Connex*t 7.x.y. For installation instructions, see the [RTI Connex](#)t Installation Guide.
- Make sure the **NDDSHOME** environment variable is set to the root directory of the *Connex*t installation (such as `/home/user/rti_connex_dds-7.x.y` or `C:\Program Files\rti_connex_dds-7.x.y`). To confirm, type this at a command prompt:

```
echo %NDDSHOME%
```

- To become familiar with *Connex*t and the build process, follow the hands-on exercises in the [RTI Connex](#)t 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 5 Linux Platforms on page 28](#)
 - [Chapter 9 Windows Platforms on page 91](#)
- 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 *Connex*t, you must specify:

- **NDDSHOME** environment variable
- *Connex*t header files

- *Connex*t libraries to link
- Compatible system libraries
- Compiler options

To build Java applications using *Connex*t, you must specify:

- NDDSHOME environment variable
- *Connex*t 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 *Connex*t applications on the same computer, *and* that computer is not connected to a network, you must set the environment variable `NDDSDISCOVERYPEERS` so that it will only use shared memory. For example:

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

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

2.2 Connex

 Header Files – All Platforms

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

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

Connex	Header Files
C	#include "nlds/nlds_c.h"
C++	#include "nlds/nlds_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 *Connex*t. See [1.1 Paths Mentioned in Documentation](#) on page 3.

Table 2.2 Include Paths for Compilation (All Platforms)

Connex 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 Connex Core Libraries User's Manual](#)).

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

Connex 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 *Connex* 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 *Connex* 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.

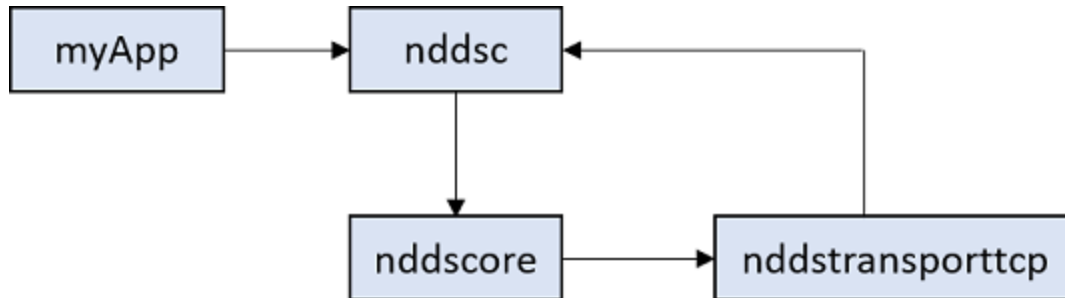
2.3.2 Mixing Static and Dynamic Libraries is not Supported

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 *Connex*-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 *Connex* application.

2.4 Building for Java Platforms

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

- *Connex* 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 *Connex* applications.

Native Libraries: *Connex* for Java is implemented using Java Native Interface (JNI), so it is necessary to provide your *Connex* 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

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

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

instead of setting the variables manually in your CMake scripts, you can call the *Connex* “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 (**FindRTIConnexDDS.cmake**) in `<NDDSHOME>/resource/cmake`. To learn about the input and output variables, see the documentation included in the script.

Chapter 3 AIX Platforms

Table 3.1 Supported AIX Platforms in Connex 7.3.0 LTS below shows the supported IBM® AIX® platforms.

Table 3.1 Supported AIX Platforms in Connex 7.3.0 LTS

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

3.1 Building Applications for AIX Platforms

See [Table 3.2 Building Instructions for AIX Architectures on the next page](#) for the compiler flags and the libraries you will need to link into your application.

Depending on which *Connex* features you want to use, you may need additional libraries; see [3.1.1 Additional Libraries for Other Features on page 12](#).

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

Table 3.2 Building Instructions for AIX Architectures

API	Library Format	Required RTI Libraries ^[a]	Required System Libraries	Required Compiler Flags
Traditional C++	Static Release	libniddscppz.a or libniddscpp2z.a libniddscz.a libniddscorez.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	N/A	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Static Debug	libniddscppzd.a or libniddscpp2zd.a libniddsczd.a libniddscorezd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a	N/A	-g -O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Dynamic Release	libniddscpp.so or libniddscpp2.so libniddsc.so libniddscore.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so	-ldl -lnsl -lm -pthread -brtl	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmshrobj -brtl -bbigtoc -qthreaded
	Dynamic Debug	libniddscppd.so or libniddscpp2d.so libniddscd.so libniddscored.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so	-ldl -lnsl -lm -pthread -brtl	-g -O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmshrobj -brtl -bbigtoc -qthreaded
C	Static Release	libniddscz.a libniddscorez.a librticonnextmsgcz.a	N/A	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Static Debug	libniddsczd.a libniddscorezd.a librticonnextmsgczd.a	N/A	-g -O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT
	Dynamic Release	libniddsc.so libniddscore.so librticonnextmsgc.so	-ldl -lnsl -lm -pthread -brtl	-O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmshrobj -brtl -bbigtoc -qthreaded
	Dynamic Debug	libniddscd.so libniddscored.so librticonnextmsgcd.so	-ldl -lnsl -lm -pthread -brtl	-g -O -q64 -qminimaltoc -DCPU=Power9 -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -G -qmshrobj -brtl -bbigtoc -qthreaded

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

3.1.1 Additional Libraries for Other Features

3.1.1.1 Libraries Required for Distributed Logger

To use the Distributed Logger APIs, link against the additional libraries in [Table 3.3 Additional Libraries for using RTI Distributed Logger below](#).

Table 3.3 Additional Libraries for using RTI Distributed Logger

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

3.1.1.2 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 *Connex* application is linked with the static release version of the *Connex* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

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

Notes:

- Memory and CPU usage is not available in monitoring data.
- If you plan to use *static* libraries, the RTI library from [Table 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

Static Release	Static Debug	Dynamic Release	Dynamic Debug
librtmonitoringz.a	librtmonitoringzd.a	librtmonitoring.so	librtmonitoringd.so

3.1.1.3 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 3.5 Additional Libraries for Zero Copy Transfer Over Shared Memory](#) below. This feature is supported in the C, traditional C++, and modern C++ programming languages.

Table 3.5 Additional Libraries for Zero Copy Transfer Over Shared Memory

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

3.1.2 How the Connex Libraries were Built

[Table 3.6](#) provides details on how the libraries were built. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

Table 3.6 Library-Creation Details for AIX Architectures

RTI Architecture	Library Format (Static & Dynamic)	Compiler Flags Used by RTI
64p9AIX7.2xclang16.1	Release	-O -q64 -qminimaltoc -D_LINUX_SOURCE_COMPAT -D_EXTENSIONS_ -DCPU=Power9 -O -fPIC -qpvc=large -qthreaded -qalias=noansi -G -qmkshrobj -brtl -DNDEBUG -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -Werror-implicit-function-declaration
	Debug	-g -O -q64 -qminimaltoc -D_LINUX_SOURCE_COMPAT -D_EXTENSIONS_ -DCPU=Power9 -O -fPIC -qpvc=large -qthreaded -qalias=noansi -G -qmkshrobj -brtl -DRTI_AIX -DRTI_UNIX -DRTI_64BIT -Werror-implicit-function-declaration

3.2 Running User Applications

Table 3.7 provides details on the environment variables that must be set at run time for an AIX architecture.

Table 3.7 Running Instructions for AIX Architectures

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

3.3 Multicast Support

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

3.4 Transports

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

3.4.1 Notes for Using Shared Memory

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

```
EXTSHM=ON
```

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

[a] See Section 3.4.1 .

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

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

The shared semaphore keys used by *Connex*t 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 *Connex*t.

3.5 Unsupported Features

These features are not supported on AIX platforms:

- Java, Python, .NET, and Modern C++ APIs
- Backtrace
- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script
- Internal setting of thread names at the operating-system level
- Remote Procedure Calls (RPCs)
- Monotonic clock

See also: [Supported Platforms tables in the RTI Connex](#)t Core Libraries Release Notes. These tables show which RTI products are supported for each platform.

3.6 Thread Configuration

See [Table 3.8 Thread Settings for AIX Platforms on the next page](#) and [Table 3.9 Thread-Priority Definitions for AIX Platforms on page 17](#).

Table 3.8 Thread Settings for AIX Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	OS default thread priority
	stack_size	192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	OS default thread priority
	stack_size	192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	4*192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	4*192*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 3.9 Thread-Priority Definitions for AIX Platforms

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

3.6.1 Changing Thread Priority

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

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

Chapter 4 Android Platforms

Table 4.1 shows the supported Android™ platforms.

Table 4.1 Supported Android Target Platforms in Connex 7.3.0 LTS

Operating System	CPU	Compiler	RTI Architecture Abbreviation
Android 12 (Advanced example generation in Code Generator not supported.)	Arm v8	Clang 12.0.8	arm64Android12clang12.0.8ndkr23b
		AdoptOpenJDK 17.0.6	

4.1 Building Applications for Android Platforms

4.1.1 Required Libraries and Compiler Flags

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

See [Table 4.2 Building Instructions for Android Architectures](#) for a list of the compiler flags and libraries you will need to link into your application.

Depending on which *Connex* features you want to use, you may need additional libraries; see [4.1.2 Additional Libraries for Other Features on page 20](#).

Additional Documentation: See the [RTI Connex Core Libraries Getting Started Guide Addendum for Android Systems](#).

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

Table 4.2 Building Instructions for Android Architectures

API	Library Format	Required RTI Libraries and JAR Files ^{[a] [b]}	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Release	libnddscore.so libnddsc.so libnddscpp.so or libnddscpp2.so librticonnextmsgcpp.so libc++_shared.so	-L\$(ANDROID_NDK_ROOT)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/aarch64-linux-android -lc++_shared	-DRTL_LINUX -DRTL_UNIX -DRTL_64BIT -DRTL_ANDROID=12
	Debug	libnddscored.so libnddscd.so libnddscppd.so or libnddscpp2d.so librticonnextmsgcppd.so libc++_shared.so		
C	Release	libnddscore.so libnddsc.so librticonnextmsgc.so	-L\$(ANDROID_NDK_ROOT)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/aarch64-linux-android/31 -llog -lc -lm	-DRTL_LINUX -DRTL_UNIX -DRTL_64BIT -DRTL_ANDROID=12
	Debug	libnddscored.so libnddscd.so librticonnextmsgcd.so		
Java	Release	When not building Apps (*.apk): nddsjava.jar rticonnextmsg.jar When building Apps (*.apk): nddsjava.jar libnddsjava.so libnddscore.so libnddsc.so rticonnextmsg.jar	N/A	None required
	Debug	When not building Apps (*.apk): nddsjavaad.jar rticonnextmsgd.jar When building Apps (*.apk): nddsjavaad.jar libnddsjavaad.so libnddscored.so libnddscd.so rticonnextmsgd.jar		

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

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

4.1.2 Additional Libraries for Other Features

4.1.2.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on all the platforms in [Table 4.1 Supported Android Target Platforms in Connex 7.3.0 LTS](#).

To use the Distributed Logger APIs, link against the additional libraries in [Table 4.3 Additional Libraries for using RTI Distributed Logger](#).

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

Table 4.3 Additional Libraries for using RTI Distributed Logger

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

4.1.2.2 Libraries Required for Monitoring

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

Table 4.4 Additional Libraries for Using Monitoring

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

4.1.2.3 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.

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

Using *Real-Time WAN Transport* requires using one of the libraries in [Table 4.5 Additional Libraries for Using RTI Real-Time WAN Transport APIs](#). Select the file appropriate for your chosen library format.

For more information, see [Enabling Real-Time WAN Transport, in the RTI Connex Core Libraries User's Manual](#).

Table 4.5 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries ^[a]
Release	libnddsrt.so
Debug	libnddsrt.d.so

4.1.2.4 Libraries Required for TCP Transport and TLS Support

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*, also link against the libraries in [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	TCP Transport Libraries ^[b]
Release	libnndstransporttcp.so
Debug	libnndstransporttcp.d.so

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

Library Format	TCP Transport Libraries ^[c]	OpenSSL Libraries ^[d]
Release	libnndstls.so	librtisslsupport.so
Debug	libnndstls.d.so	

[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-3.0.12/<architecture>/<format>/lib.

4.1.3 Target Configuration

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

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

For a raw Linux distribution, all programs are executables that are linked with the necessary *Connex* libraries (see [Table 4.1 Supported Android Target Platforms in Connex 7.3.0 LTS](#)). The build process is similar to other Linux variants, see [5.1 Building Applications for Linux Platforms on page 30](#).

4.1.4 'Release' and 'Debug' Terminology

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

4.1.5 How the Connex Libraries were Built

[Table 4.8 Library-Creation Details for Android Architectures on the next page](#) provides details on how the libraries were built. This table is provided strictly for informational purposes; you do not need to use these parameters to compile your application. You may find this information useful if you are involved in any in-depth debugging.

The details for building *user* applications is in [4.1 Building Applications for Android Platforms on page 18](#).

Table 4.8 Library-Creation Details for Android Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
arm64Android12clang12.0.8ndkr23b when not using Java	Release	<pre>--target=aarch64-none-linux-android31 --sysroot=/opt/toolchains/arm64Android12clang12.0.8ndkr23b/android-ndk-r23b/toolchains/llvm/prebuilt/linux-x86_64/sysroot -DLINUX -DPtrIntType=long DRTI_64BIT -DRTI_ANDROID=12 -DRTI_ENDIAN_LITTLE -DRTI_LINUX -DRTI_LINUX26 -DRTI_MONITORING_ARCHITECTURE -DRTI_MULTICAST -DRTI_NDDS_VERSION_MAJOR=6 -DRTI_NDDS_VERSION_MINOR=1 -DRTI_NDDS_VERSION_RELEASE=1 -DRTI_NDDS_VERSION_REVISION=0 -DRTI_OPENSSL_ARCHITECTURE -DRTI_POSIX_SEMAPHORES -DRTI_POSIX_THREADS -DRTI_PRECONDITION_TEST -DRTI_RTISQLITE3_ARCHITECTURE -DRTI_SECURITY_ARCHITECTURE -DRTI_SHARED_BUILD -DRTI_UNIX -DRTI_X64CPU -O0 -Wall -Wno-unknown-pragmas -Wno-deprecated-declarations -Wno-macro-redefined -Wno-tautological-pointer-compare -Wno-logical-not-parentheses -Wno-constant-conversion -Wno-return-type-c-linkage -Wno-deprecated-register -Wno-tautological-constant-out-of-range-compare -Wno-enum-conversion -Wno-format-security -Wno-switch-bool -Wno-instantiation-after-specialization -Wno-exceptions -Wno-non-literal-null-conversion -Wstrict-prototypes -Wunused-parameter -funwind-tables -no-canonical-prefixes -fexceptions -DNDEBUG -fPIC -Werror-implicit-function-declaration</pre>
	Debug	<pre>--target=aarch64-none-linux-android31 --sysroot=/opt/toolchains/arm64Android12clang12.0.8ndkr23b/android-ndk-r23b/toolchains/llvm/prebuilt/linux-x86_64/sysroot -DLINUX -DRTI_ANDROID=12 -DPtrIntType=long DRTI_64BIT -DRTI_ANDROID=12 -DRTI_ENDIAN_LITTLE -DRTI_LINUX -DRTI_LINUX26 -DRTI_MONITORING_ARCHITECTURE -DRTI_MULTICAST -DRTI_NDDS_VERSION_MAJOR=6 -DRTI_NDDS_VERSION_MINOR=1 -DRTI_NDDS_VERSION_RELEASE=1 -DRTI_NDDS_VERSION_REVISION=0 -DRTI_OPENSSL_ARCHITECTURE -DRTI_POSIX_SEMAPHORES -DRTI_POSIX_THREADS -DRTI_PRECONDITION_TEST -DRTI_RTISQLITE3_ARCHITECTURE -DRTI_SECURITY_ARCHITECTURE -DRTI_SHARED_BUILD -DRTI_UNIX -DRTI_X64CPU -O0 -Wall -Wno-unknown-pragmas -Wno-deprecated-declarations -Wno-macro-redefined -Wno-tautological-pointer-compare -Wno-logical-not-parentheses -Wno-constant-conversion -Wno-return-type-c-linkage -Wno-deprecated-register -Wno-tautological-constant-out-of-range-compare -Wno-enum-conversion -Wno-format-security -Wno-switch-bool -Wno-instantiation-after-specialization -Wno-exceptions -Wno-non-literal-null-conversion -Wstrict-prototypes -Wunused-parameter -funwind-tables -no-canonical-prefixes -fexceptions -fPIC -Werror-implicit-function-declaration</pre>
All supported Android architectures for Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g

4.2 Running Your Applications

For the environment variables that must be set at run time, see [Table 4.9 Running Instructions for Android Architectures](#).

Table 4.9 Running Instructions for Android Architectures

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

Table 4.9 Running Instructions for Android Architectures

RTI Architecture	Library Format	Required Environment Variables
All supported Android architectures when using Java	App (*.apk)	None
	Dex	LD_LIBRARY_PATH=\$LD_LIBRARY_PATH: <path-to-ndds-libs> CLASSPATH=<path-to-dex>/classes.dex

4.3 Support for Modern C++ API

Connex 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 Connex Core Libraries User's Manual](#).

4.4 Multicast Support

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

4.5 Transports

- **Shared memory:** Not supported for this release. For a consumer device, shared memory communication between Apps is often not desirable.
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported (no traffic class support).
- **TCP/IPv4:** Supported.

4.6 Unsupported Features

These features are not supported on Android platforms:

- .NET and Python APIs
- Backtrace

- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script
- Setting of thread names by *Connex*t at the operating-system level
- Using DDS_WireProtocolQosPolicyAutoKind's RTPS_AUTO_ID_FROM_MAC to calculate the GUID prefix is not supported.
- Zero Copy Transfer Over Shared Memory

See also: [Supported Platforms tables in the RTI Connex](#)t Core Libraries Release Notes. These tables show which RTI products are supported for each platform.

4.7 Monotonic Clock Support

The monotonic clock (described in [Configuring the Clock per DomainParticipant, in the RTI Connex](#)t Core Libraries User's Manual) is supported on all Android platforms.

4.8 Thread Configuration

See [Table 4.10 Thread Settings for Android Platforms](#) and [Table 4.11 Thread-Priority Definitions for Android Platforms](#).

Table 4.10 Thread Settings for Android Platforms

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

Table 4.10 Thread Settings for Android Platforms

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

Table 4.11 Thread-Priority Definitions for Android Platforms

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

4.9 Support for Remote Procedure Calls (RPCs)

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

See [Remote Procedure Calls, in the RTI Connex Core Libraries User's Manual](#).

4.10 Third-Party Software Versions used for Android 12 Development and Testing

Table 4.12 Third-Party Software Versions Used for Android 12 Development and Testing

Third-Party Software	Version
Android Studio	Bumblebee 2021.1.1 Patch 3 (2020.3.1 or higher)
Android Build Tools	33
Android Command-Line Tools	6.0
Android SDK Platforms-Tools	33
Android API Level	31
Android NDK	NDK 23b LTS (23.1.7779620)
Gradle Plugin	7.3.0
Gradle	7.5.1

Chapter 5 Linux Platforms

Table 5.1 shows the supported Linux® platforms.

Table 5.1 Supported Linux Platforms in Connex 7.3.0 LTS

OS	Version	CPU	GLIBC	GLIBCXX	Toolchain	RTI Architecture Abbreviation
Linux (Intel)	Red Hat Enterprise Linux 8, 9 Ubuntu 18.04 LTS, 20.04 LTS, 22.04 LTS	x64	2.27	6.0.25	gcc 7.3.0 or AdoptOpenJDK 17.0.6	x64Linux4gcc7.3.0
	Ubuntu 22.04 LTS	x64	2.35	N/A	clang 15.0.1 [a] or AdoptOpenJDK 17.0.6	x64Linux5Unreal5.2clang15 [b]
	Red Hat Enterprise Linux 8 Ubuntu 18.04 LTS, 20.04 LTS (Only for use with <i>Connex TSS</i>)	x64	2.27	6.0.25	gcc 7.3.0	x64Linux4gcc7.3.0FACE_GP [c]

[a] Included with Unreal Engine® 5.2.1

[b] Target libraries for Unreal Engine 5.2.

[c] FACE platforms do not support the Request/Reply API, DDS Spy, or DDS Ping.

Table 5.1 Supported Linux Platforms in Connex 7.3.0 LTS

OS	Version	CPU	GLIBC	GLIBCXX	Toolchain	RTI Architecture Abbreviation
Linux (Arm)	Ubuntu 18.04 LTS	Arm v7	2.27	6.0.25	gcc 7.5.0 or AdoptOpenJDK 17.0.6	armv7Linux4gcc7.5.0 [a]
	Ubuntu 18.04 LTS, 22.04 LTS	Arm v8	2.27	6.0.25	gcc 7.3.0 or AdoptOpenJDK 17.0.6	armv8Linux4gcc7.3.0
The following rows show custom-supported target platforms, only available on demand. Please contact your RTI sales representative or sales@rti.com for more information on using these platforms:						
Linux (Intel)	Red Hat Enterprise Linux 7, 7.3, 7.5, 7.6 CentOS 7.0	x64	2.17	6.0.19	gcc 4.8.2 or AdoptOpenJDK 17.0.6	x64Linux3gcc4.8.2 (<i>Custom support</i>)
		x86	2.17	6.0.19	gcc 4.8.2	i86Linux3gcc4.8.2 (<i>Custom support</i>)
	RedHawk Linux 8.4.1	x64	2.28	6.0.19	gcc 8.5.0 or AdoptOpenJDK 17.0.6	x64RedHawk8.4gcc8.5.0 (<i>Custom support</i>)
		x86	2.17	6.0.19	gcc 8.5.0	i86RedHawk8.4gcc8.5.0 (<i>Custom support</i>)
Linux (Arm)	TI Linux 8.2.0.3	Arm v8	2.30	6.0.28	gcc 9.2.1 or AdoptOpenJDK 17.0.6	armv8Linux-armgcc9.2.1 (<i>Custom support</i>)

(*Custom support*) means this is a custom-supported target architecture, only available on demand. Please contact your RTI sales representative or sales@rti.com for more information.

[a] These libraries require a hardware FPU in the processor and are compatible with systems that have hard-float libc. See [5.1 Building Applications for Linux Platforms on the next page](#) for compiler flag details.

5.1 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:

- *Connex* 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 Connex 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 *Connex* installation (such as `/home/user/rti_connex_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>
```

5.1.1 Required Libraries and Compiler Flags

To compile a *Connex* application of any complexity, either modify the auto-generated makefile created by running *RTI Code Generator* or write your own makefile. See [5.1 Building Applications for Linux Platforms above](#) for required compiler flags.

[Table 5.2 Building Instructions for Linux Architectures](#) lists the compiler flags and libraries you will need to link into your application.

Depending on which *Connex* features you want to use, you may need additional libraries; see [5.1.2 Additional Libraries for Other Features on page 33](#)

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

Table 5.2 Building Instructions for Linux Architectures

API	Library Format	Required RTI Libraries or Jar Files ^[a] ^[b]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs) (Modern C++ API is not available for x64Linux5Unreal5.2clang15)	Static Release	libnndscorez.a libnndscz.a libnndscppz.a or libnndscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-ldl -lm -lpthread -lrt For Ubuntu 18.04 LTS on Arm v7, also add: -latomic	For 64-bit architectures: -DRTI_LINUX -DRTI_UNIX -m64 For Unreal Engine 5.2.1, also add: -stdlib=libc++ -fno-rtti 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 5.13 Backtrace Support on page 46)
	Static Debug	libnndscorezd.a libnndsczd.a libnndscppzd.a or libnndscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnndscore.so libnndsc.so libnndscpp.so or libnndscpp2.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so		
	Dynamic Debug	libnndscored.so libnndscd.so libnndscppd.so or libnndscpp2d.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so		

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

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

Table 5.2 Building Instructions for Linux Architectures

API	Library Format	Required RTI Libraries or Jar Files ^[a] ^[b]	Required System Libraries	Required Compiler Flags
C	Static Release	libnndscorez.a libnndscz.a librticonnextmsgcz.a	-ldl -lm -lpthread -lrt For Ubuntu 18.04 LTS on Arm v7, also add: -latomic	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 5.13 Backtrace Support on page 46)
	Static De- bug	libnndscorezd.a libnndsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnndscore.so libnndsc.so librticonnextmsgc.so		
	Dynamic Debug	libnndscored.so libnndscd.so librticonnextmsgcd.so		
Java	Release	nndsjava.jar rticonnextmsg.jar	N/A	None required
	Debug	nndsjava.d.jar rticonnextmsg.d.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.

[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.

5.1.2 Additional Libraries for Other Features

5.1.2.1 Libraries Required for Distributed Logger

To see which platforms support *RTI Distributed Logger*, refer to the [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#).

To use the Distributed Logger APIs, link against the additional libraries in [Table 5.3 Additional Libraries for using RTI Distributed Logger](#).

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	librtidlczd.a	librtidlc.so	librtidcd.so
	librtidlcppz.a	librtidlcppzd.a	librtidlcpp.so	librtidlcppd.so
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

5.1.2.2 Libraries Required for Monitoring

To see which platforms support Monitoring, refer to the [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#).

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

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

Note: If you plan to use *static* libraries, the RTI library in [Table 5.4 Additional Libraries for Using Monitoring on the next page](#) must appear *first* in the list of libraries to be linked.

Table 5.4 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries ^[a]
Dynamic Release	librtmonitoring.so
Dynamic Debug	librtmonitoringd.so
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

5.1.2.3 Libraries Required for Real-Time WAN Transport

To see which platforms support *Real-Time WAN Transport*, refer to the [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#).

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 5.5 Additional Libraries for Using Real-Time WAN Transport APIs](#). Select the file appropriate for your chosen library format.

For more information, see [Enabling Real-Time WAN Transport, in the RTI Connex Core Libraries User's Manual](#).

Table 5.5 Additional Libraries for Using Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries ^[b]
Dynamic Release	libnndsrwt.so
Dynamic Debug	libnndsrwtd.so
Static Release	libnndsrwtz.a
Static Debug	libnndsrwtzd.a

5.1.2.4 Libraries Required for TCP Transport and TLS Support

To see which platforms support *TLS Support*, refer to the [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#). For information on which platforms support the TCP transport, see [5.7 Transports on page 42](#).

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

Library Format	RTI TCP Transport Libraries ^[a]
Dynamic Release	libniddstransporttcp.so
Dynamic Debug	libniddstransporttcpd.so
Static Release	libniddstransporttcpz.a
Static Debug	libniddstransporttcpzd.a

If you are using *RTI TLS Support*, see [Table 5.7 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 Connex*®. If you choose to use *TLS Support*, it must be installed on top of a *Connex* installation with the same version number; it can only be used on architectures that support TCP transport.

Table 5.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries ^[b]	OpenSSL Libraries ^[c]
Dynamic Release	libniddstls.so	libssl.so libcrypto.so
Dynamic Debug	libniddstlsd.so	
Static Release	libniddstlsz.a	
Static Debug	libniddstlszd.a	

5.1.2.5 Libraries Required for Zero Copy Transfer Over Shared Memory

The Zero Copy Transfer Over Shared Memory feature is supported on all the platforms in [Table 5.1 Supported Linux Platforms in Connex 7.3.0 LTS on page 28](#).

To use this feature, link against the additional library in [Table 5.8 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-3.0.12/<architecture>/<format>/lib.

Table 5.8 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries ^[a]
Dynamic Release	libniddsmetp.so
Dynamic Debug	libniddsmetpd.so
Static Release	libniddsmetpz.a
Static Debug	libniddsmetpzd.a

5.1.3 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 *Connex* 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

5.1.3.1 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 *Connex* 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 *Connex*, 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.

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.

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

Table 5.9 Compatibility Ranges

Library Name	CPU	Minimum GLIBC	GLIBCXX Range
x64Linux3gcc4.8.2	x64	2.17	6.0.19 <= 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

5.1.3.2 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.

5.1.3.3 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.

5.1.4 How the Connex Libraries were Built

Table 5.10 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 5.10 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 -Werror=implicit-function-declaration
	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 -Werror=implicit-function-declaration
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" -Werror=implicit-function-declaration
	Static Debug	-O0 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="\armv8Linux4gcc7.3.0" -Werror=implicit-function-declaration
	Dynamic Release	-O -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="\armv8Linux4gcc7.3.0" -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="\armv8Linux4gcc7.3.0" -Werror=implicit-function-declaration
armv8Linux-armgcc9.2.1	Static Release	-DLINUX -DPtrIntType=long -DTARGET="\armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -Werror=implicit-function-declaration
	Static Debug	-DLINUX -DPtrIntType=long -DTARGET="\armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -Werror=implicit-function-declaration
	Dynamic Release	-DLINUX -DPtrIntType=long -DTARGET="\armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O -DNDEBUG -fPIC -Werror=implicit-function-declaration
	Dynamic Debug	-DLINUX -DPtrIntType=long -DTARGET="\armv8Linux-armgcc9.2.1" -O -Wall -Wno-unknown-pragmas -feliminate-unused-debug-types -fno-omit-frame-pointer -funwind-tables -O0 -g -fPIC -Werror=implicit-function-declaration

Table 5.10 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
i86Linux3gcc4.8.2	Static Release	-O -m32 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="i86Linux3gcc4.8.2" -Werror=implicit-function-declaration
	Static Debug	-O0 -m32 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="i86Linux3gcc4.8.2" -Werror=implicit-function-declaration
	Dynamic Release	-O -m32 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="i86Linux3gcc4.8.2" -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -m32 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="i86Linux3gcc4.8.2" -Werror=implicit-function-declaration
i86RedHawk8.4gcc8.5.0	Static Release	-m32 -Wall -Wno-unknown-pragmas -O -DNDEBUG -Werror=implicit-function-declaration
	Static Debug	-m32 -Wall -Wno-unknown-pragmas -O0 -g -Werror=implicit-function-declaration
	Dynamic Release	-m32 -Wall -Wno-unknown-pragmas -O -DNDEBUG -fPIC -Werror=implicit-function-declaration
	Dynamic Debug	-m32 -Wall -Wno-unknown-pragmas -O0 -g -fPIC -Werror=implicit-function-declaration
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" -Werror=implicit-function-declaration
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux3gcc4.8.2" -Werror=implicit-function-declaration
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux3gcc4.8.2" -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux3gcc4.8.2" -Werror=implicit-function-declaration
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" -Werror=implicit-function-declaration
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc7.3.0" -Werror=implicit-function-declaration
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc7.3.0" -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc7.3.0" -Werror=implicit-function-declaration

Table 5.10 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Linux4gcc7.3.0FACE_GP	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc7.3.0FACE_GP" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL -Werror=implicit-function-declaration
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc7.3.0FACE_GP" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL -Werror=implicit-function-declaration
	Dynamic Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc7.3.0FACE_GP" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc7.3.0FACE_GP" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL -Werror=implicit-function-declaration
x64Linux5Unreal5.2clang15	Release (static and dynamic)	--sysroot=<unreal_install_path>/Engine/Extras/ThirdPartyNotUE/SDKs/HostLinux/ Linux_x64/v21_clang-15.0.1-centos7/x86_64-unknown-linux-gnu -O0 -fno-omit-frame-pointer -DNDEBUG -fPIC -Werror=implicit-function-declaration -fuse-lld=lld
	Debug (static and dynamic)	--sysroot=<unreal_install_path>/Engine/Extras/ThirdPartyNotUE/SDKs/HostLinux/ Linux_x64/v21_clang-15.0.1-centos7/x86_64-unknown-linux-gnu -O0 -fno-omit-frame-pointer -g -fPIC -Werror=implicit-function-declaration -fuse-lld=lld
x64RedHawk8.4gcc8.5.0	Static Release	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -Werror=implicit-function-declaration
	Static Debug	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -Werror=implicit-function-declaration
	Dynamic Release	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -Werror=implicit-function-declaration
	Dynamic Debug	-m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O0 -g -fPIC -Werror=implicit-function-declaration
All supported Linux architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

5.2 Running Your Applications

For the environment variables that must be set at run time, see [Table 5.11 Running Instructions for Linux Architectures below](#).

Table 5.11 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 <u>not</u> using Java	Static (Release & Debug)	None required
	Dynamic (Release & Debug)	LD_LIBRARY_PATH= \${NDDSHOME}/lib/<architecture>: \${LD_LIBRARY_PATH}

5.3 Support for the Modern C++ API

*Connex*t 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 Connex Core Libraries User's Manual](#).

The Modern C++ API is available for all supported Linux platforms except the POSIX-compliant architectures that end with "FACE_GP" and the x64Linux5Unreal5.2clang15 architecture (because it requires Run-Time Type Information, which is not supported by the Unreal Engine).

5.4 Support for the .NET (C#) API

The C# API is supported on Intel x64, Arm v7, and Arm v8 CPUs. For more information on .NET, see the [Connex C# API Reference](#).

5.5 Support for the Python API

The Python API is supported for Python 3.6 - 3.12, on Intel x64 and Arm v8 CPUs. For more information, see the [Connex Python API Reference](#).

5.6 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.

5.7 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, except POSIX-compliant architectures that end with "FACE_GP". This is *not* a built-in transport.

5.7.1 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 *Connex*t are in the range of 0x400000. For example:

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

The shared semaphore keys used by *Connex*t 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 *Connex*t.

5.8 Limitations of FACE Architectures

This section describes limitations when using a FACE architecture. This is a POSIX-compliant architecture, available with *RTI Connex*t TSS:

- x64Linux4gcc7.3.0FACE_GP

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

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

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

- Java, Python, .NET, and Modern C++ APIs
- Backtrace
- Cmake Find package
- Distributed Logger
- Durable writer history and durable reader state
- Modern C++
- Monitoring
- Real-time clock
- Request/Reply communication pattern
- Remote Procedure Calls
- Setting thread names by *Connex* at the operating-system level
- RTI DDS Ping and Spy

See also: [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#). These tables show which RTI products are supported for each platform.

5.9 Monotonic Clock Support

The monotonic clock (described in [Configuring the Clock per DomainParticipant, in the RTI Connex Core Libraries User's Manual](#)) is supported on all Linux platforms.

5.10 Thread Configuration

[Table 5.12 Thread Settings for Linux Platforms on the next page](#) lists the thread settings for Linux platforms.

See also: [Table 5.13 Thread-Priority Definitions for Linux Platforms on page 45](#) and [Table 5.14 Thread Kinds for Linux Platforms on page 45](#).

5.10.1 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 except x64Linux5Unreal5.2clang15.

Table 5.12 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
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 5.13 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 5.14 Thread Kinds for Linux Platforms

Thread Kinds	Operating-System Configuration ^a
DDS_THREAD_SETTINGS_FLOATING_POINT	N/A
DDS_THREAD_SETTINGS_STUDIO	N/A
DDS_THREAD_SETTINGS_REALTIME_PRIORITY	Set schedule policy to SCHED_FIFO
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE	N/A

5.10.2 Using REALTIME_PRIORITY

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

^aSee the Linux programmer's manuals for more information.

5.11 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:

- x64RedHawk8.4gcc8.5.0
- i86RedHawk8.4gcc8.5.0
- x64Linux4gcc7.3.0FACE_GP
- armv8Linux-armgcc9.2.1

5.12 Support for 'Find Package' CMake Script

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

- x64Linux3gcc4.8.2
- x64Linux4gcc7.3.0
- x64Linux5Unreal5.2clang15

For information on using this script, see [2.5 Building with CMake on page 8](#)

5.13 Backtrace Support

Backtrace is supported on all Linux platforms except POSIX-compliant architectures that end with "FACE_GP".

- 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 5.2 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 Connexx Core Libraries User's Manual](#).

5.14 Support for Remote Procedure Calls (RPC)

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

- armv8Linux-armgcc9.2.1
- armv7Linux4gcc7.5.0
- armv8Linux4gcc7.3.0
- i86RedHawk8.4gcc8.5.0
- x64Linux4gcc7.3.0
- x64RedHawk8.4gcc8.5.0

See [Remote Procedure Calls, in the RTI Connex Core Libraries User's Manual](#).

Chapter 6 macOS Platforms

[Table 6.1 Supported macOS Platforms in Connex 7.3.0 LTS](#) lists the architectures supported on macOS® operating systems.

Table 6.1 Supported macOS Platforms in Connex 7.3.0 LTS

Operating System	CPU	Compiler	RTI Architecture Abbreviation
macOS 11, 12 , and 13 (host and target)	x64	clang 12.0, 13.0, 14.0 or AdoptOpenJDK 17.0.6	x64Darwin20clang12.0
	Arm v8	clang 12.0, 13.0, 14.0 or AdoptOpenJDK 17.0.6	arm64Darwin20clang12.0

6.1 Building Applications for macOS Platforms

[Table 6.2 Building Instructions for macOS Architectures](#) lists the compiler flags and libraries you will need to link into your application. Depending on which *Connex* features you want to use, you may need additional libraries; see [6.1.1 Additional Libraries for Other Features on page 51](#).

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 macOS Architectures

API	Library Format	Required RTI Libraries [a] [b]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnddscores.a libnddscz.a libnddscppz.a or libnddscpp2z.a librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-ldl -lm -lpthread	For x64 architectures: -dynamic -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.

^bThe *Connex* C/C++ libraries are in `<NDDSHOME>/lib/<architecture>/`.

`<NDDSHOME>` is where *Connex* is installed, see [1.1 Paths Mentioned in Documentation on page 3](#)

Table 6.2 Building Instructions for macOS Architectures

API	Library Format	Required RTI Libraries [a] [b]	Required System Libraries	Required Compiler Flags
C	Static Release	libnddscorez.a libnddscz.a librticonnextmsgcz.a	-ldl -lm -lpthread	For x64 architectures: -dynamic -single_module -DRTI_UNIX -DRTI_DARWIN -DRTI_64BIT For Arm v8 architectures: -DRTI_UNIX -DRTI_DARWIN
	Static Debug	libnddscorezd.a libnddsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnddscore.dylib libnddsc.dylib librticonnextmsgc.dylib		
	Dynamic Debug	libnddscored.dylib libnddscd.dylib librticonnextmsgcd.dylib		
Java	Release	nddsjava.jar rticonnextmsg.jar	N/A	None required
	Debug	nddsjavad.jar rticonnextmsgd.jar		

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

^bThe *Connex* C/C++ libraries are in <NDDSHOME>/lib/<architecture>/.

<NDDSHOME> is where *Connex* is installed, see [1.1 Paths Mentioned in Documentation on page 3](#)

6.1.1 Additional Libraries for Other Features

6.1.1.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on macOS platforms. [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		Dynamic	
	Release	Debug	Release	Debug
C++ (Traditional API)	librtidlcz.a librtidlcppz.a	librtidlczd.a librtidlcppzd.a	librtidlc.dylib librtidlcpp.dylib	librtidlcd.dylib librtidlcppd.dylib
C	librtidlcz.a	librtidlczd.a	librtidlc.dylib	librtidlcd.dylib
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

6.1.1.2 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 *Connex*t application is linked with the static release version of the *Connex*t libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

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

Note: If you are plan to use *static* libraries, the RTI library in [Table 6.4 Additional Libraries for Using Monitoring](#) must appear *first* in the list of libraries to be linked.

Table 6.4 Additional Libraries for Using Monitoring

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

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

Table 6.4 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries ^[a]
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

6.1.1.3 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 [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 [Enabling Real-Time WAN Transport, in the RTI Connex Core Libraries User's Manual](#).

Table 6.5 Additional Libraries for Using Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries ^[b]
Dynamic Release	libnndsrwt.dylib
Dynamic Debug	libnndsrwtd.dylib
Static Release	libnndsrwtz.a
Static Debug	libnndsrwtzd.a

6.1.1.4 Libraries Required for TCP Transport

To use the TCP Transport APIs, link against the additional libraries in [Table 6.6 Additional Libraries for using RTI TCP Transport APIs](#). If you are using *RTI TLS Support*, see [Table 6.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled](#). Select the files appropriate for your chosen library format.

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

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

Table 6.6 Additional Libraries for using RTI TCP Transport APIs

Library Format	RTI TCP Transport Libraries ^[a]
Dynamic Release	libniddstransporttcp.dylib
Dynamic Debug	libniddstransporttcpd.dylib
Static Release	libniddstransporttcpz.a
Static Debug	libniddstransporttcpzd.a

Table 6.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries ^[b]	OpenSSL Libraries ^[c]
Dynamic Release	libniddstls.dylib	libssl.dylib libcrypto.dylib
Dynamic Debug	libniddstlsd.dylib	
Static Release	libniddstlsz.a	libssl.a libcryptoz.a
Static Debug	libniddstlszd.a	

6.1.1.5 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 6.8 Additional Libraries for Zero Copy Transfer Over Shared Memory](#) .

Table 6.8 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Library
Dynamic Release	libniddsmetp.dylib
Dynamic Debug	libniddsmetpd.dylib
Static Release	libniddsmetpz.a
Static Debug	libniddsmetpzd.a

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

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

[c] OpenSSL libraries are in <NDDSHOME>/third_party/openssl-3.0.12/<architecture>/<format>/lib..

6.1.2 How the Connext Libraries were Built

Table 6.9 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 6.9 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 -Werror=implicit-function-declaration
	Debug	-Dunix -O0 -Wall -Wno-unknown-pragmas -Wno-trigraphs -Wmissing-field-initializers -Wuninitialized -O0 -g -fPIC -Werror=implicit-function-declaration
x64Darwin20clang12.0	Release	-arch x86_64 -Wno-trigraphs -fpascal-strings -fasm-blocks -O -Wall -Wno-unknown-pragmas -DPtrIntType=long -DTARGET=\"x64Darwin20clang12.0\" -DNDEBUG -Werror=implicit-function-declaration
	Debug	-arch x86_64 -Wno-trigraphs -fpascal-strings -fasm-blocks -g -O -Wall -Wno-unknown-pragmas -DPtrIntType=long -DTARGET=\"x64Darwin20clang12.0\" -Werror=implicit-function-declaration
arm64Darwin20clang12.0 and x64Darwin20clang12.0 for Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g

6.2 Running User Applications

[Table 6.10 Running Instructions for macOS Architectures](#) provides details on the environment variables that must be set at run time for a macOS architecture.

Table 6.10 Running Instructions for macOS Architectures

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

6.3 Support for the Modern C++ API

Connex 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 Connex Core Libraries User's Manual](#).

6.4 Support for the .NET (C#) API

The C# API is supported on macOS platforms with Intel and Apple silicon (Arm v8) CPUs. For more information on .NET, see the [Connex C# API Reference](#).

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

6.5 Support for the Python API

The Python API is supported on macOS platforms Intel and Apple silicon (Arm v8) CPUs. For more information, see the [Connex Python API Reference](#).

6.6 Multicast Support

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

6.7 Transports

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

6.8 Unsupported Features

These features are not supported on macOS platforms:

- Controlling CPU Core Affinity
- Monotonic clock

See also: [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#). These tables show which RTI products are supported for each platform.

6.9 System Integrity Protection (SIP)

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

6.9.1 SIP and Java Applications

If you run *Connex* 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 *Connex* 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 *Connex* libraries. This is automatically performed by the scripts to run applications generated by the *RTI Code Generator*. However, if you are manually running your *Connex* 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 *Connex* 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 *Connex* infrastructure.

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

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

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

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

For more on transport plugins, see [6.1.1.4 Libraries Required for TCP Transport on page 52](#).

6.9.2 SIP and Connex 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 work around this issue, *Connex* 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>
```

6.10 Thread Configuration

See [Table 6.11 Thread Settings for macOS Platforms](#) and [Table 6.12 Thread-Priority Definitions for macOS Platforms](#).

Table 6.11 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_STUDIO
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 6.11 Thread Settings for macOS Platforms

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

Table 6.12 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	

6.11 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on macOS platforms. For information on using this script, see [2.5 Building with CMake on page 8](#).

6.12 Backtrace Support

Backtrace is supported on macOS platforms and is configured out of the box. See [Logging a Backtrace for Failures, in the RTI Connex Core Libraries User's Manual](#).

6.13 Resolving NDDUtility_sleep() Issues

When running on a macOS system, you may experience timing issues in your calls to `NDDUtility_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 *Connex* application? Suppose you send samples from your publisher at a 5 ms rate, using `NDDUtility_sleep()` to calculate that wait time. You have a subscriber with a deadline set to 6 ms. The timer coalescing feature could make your sleep last much longer than 5-6 ms, so when the next sample reaches the subscriber, the deadline period has expired and you will experience missed samples.

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

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

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

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

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

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

The reply should be:

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

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

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

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

6.14 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, in the RTI Connex Core Libraries User's Manual](#).

Chapter 7 QNX Platforms

Table 7.1 Supported QNX Platforms for Connex 7.3.0 LTS lists the architectures supported on QNX operating systems.^[a]

Table 7.1 Supported QNX Platforms for Connex 7.3.0 LTS

Operating System	CPU	Compiler	RTI Architecture
QNX Neutrino 7.1	Arm v8	qcc_gpp 8.3.0 (GNU C++ library)	armv8QNX7.1qcc_gpp8.3.0
	x64	qcc_cxx 8.3.0 (LLVM C++ library)	x64QNX7.1qcc_cxx8.3.0
<p>The following rows show custom-supported target platforms, only available on demand. Please contact your RTI sales representative or sales@rti.com for more information on using these platforms:</p>			
QNX Neutrino 7.0.4 [b]	Arm v7	qcc_cxx 5.4.0	armv7QNX7.0.0qcc_cxx5.4.0 [c] <i>(Custom support)</i>
	Arm v8	qcc_cxx 5.4.0 (LLVM C++ library)	armv8QNX7.0.0qcc_cxx5.4.0 <i>(Custom support)</i>
	x64	qcc_gpp 5.4.0 (GNU C++ library)	x64QNX7.0.0qcc_gpp5.4.0 <i>(Custom support)</i>
QNX Neutrino 7.1	Arm v8	qcc_cxx 8.3.0 (LLVM C++ library)	armv8QNX7.1qcc_cxx8.3.0 <i>(Custom support)</i>
QNX for Safety (QOS 2.2)	Arm v8	qcc_cxx 8.3.0 (LLVM C++ library)	armv8QOS2.2qcc_cxx8.3.0 <i>(Custom support)</i>

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

[b] Tested with QNX 7.0.0 kernel.

[c] Requires a hardware FPU in the processor and is compatible with systems that have hard-float libc.

7.1 Building Applications for QNX Platforms

The libraries on Arm 7 CPUs require a hardware FPU in the processor and are compatible with systems that have hard-float libc. See [Table 7.9 Library-Creation Details for QNX Architectures](#) for compiler flag details.

[Table 7.2 Building Instructions for QNX Architectures](#) lists the libraries you will need to link into your application.

Depending on which *Connex*t features you want to use, you may need additional libraries; see [7.1.2 Additional Libraries for Other Features on page 64](#).

Additional Documentation: You should also review the QNX chapter of the [RTI Connex 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 7.2 Building Instructions for QNX Architectures

API	Library Format	RTI Libraries ^[a] ^[b]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libniddscorez.a libniddscz.a libniddscppz.a or libniddscpp2z.a librtconnectmsgcppz.a or librtconnectmsgcpp2z.a	-lm -lsocket	-DRTL_QNX
	Static Debug	libniddscorezd.a libniddsczd.a libniddscppzd.a or libniddscpp2zd.a librtconnectmsgcppzd.a or librtconnectmsgcpp2zd.a		
	Dynamic Release	libniddscore.so libniddsc.so libniddscpp.so or libniddscpp2.so librtconnectmsgcpp.so or librtconnectmsgcpp2.so		
	Dynamic Debug	libniddscored.so libniddscd.so libniddscppd.so or libniddscpp2d.so librtconnectmsgcppd.so or librtconnectmsgcpp2d.so		
C	Static Release	libniddscorez.a libniddscz.a librtconnectmsgcz.a	-lm -lsocket	-DRTL_QNX
	Static Debug	libniddscorezd.a libniddsczd.a librtconnectmsgczd.a		
	Dynamic Release	libniddscore.so libniddsc.so librtconnectmsgc.so		
	Dynamic Debug	libniddscored.so libniddscd.so librtconnectmsgcd.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>.

7.1.1 Required Change for Building with C++ Libraries

The C++ libraries for QNX platforms are built *without* the **-frtti** 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 **-frtti**, but doing so will not cause a problem.

7.1.2 Additional Libraries for Other Features

7.1.2.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on all QNX platforms.

[Table 7.3 Additional Libraries for using RTI Distributed Logger](#) lists the additional libraries you will need in order to use *Distributed Logger*.

Table 7.3 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	librtidcd.so librtidlcppd.so

7.1.2.2 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 *Connex* application is linked with the static release version of the *Connex* libraries, you will need to also use the static release version of the monitoring library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

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

Notes:

- To use *static* libraries: the RTI library from [Table 7.4 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 7.4 Additional Libraries for Using Monitoring

Library Format	Monitoring Libraries ^[a]
Dynamic Release	librtmonitoring.so
Dynamic Debug	librtmonitoringd.so
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

7.1.2.3 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 7.5 Additional Libraries for Using RTI Real-Time WAN Transport APIs](#). Select the file appropriate for your chosen library format.

For more information, see [Enabling Real-Time WAN Transport, in the RTI Connex Core Libraries User's Manual](#).

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

Table 7.5 Additional Libraries for Using RTI Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries ^[a]
Dynamic Release	libnddsrt.so
Dynamic Debug	libnddsrt.d.so
Static Release	libnddsrtz.a
Static Debug	libnddsrtzd.a

7.1.2.4 Libraries Required for TCP Transport APIs and TLS Support

To use the TCP Transport APIs, link against the additional libraries in [Table 7.6 Additional Libraries for using RTI TCP Transport APIs](#).

Note: Not all platforms support the TCP Transport - see [7.5 Transports on page 71](#).

Table 7.6 Additional Libraries for using RTI TCP Transport APIs

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

If you are using *RTI TLS Support*, also see [Table 7.7 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 7.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

Library Format	RTI TLS Libraries ^[a]	OpenSSL Libraries ^[b]
Dynamic Release	libnndstls.so	libssl.so libcrypto.so
Dynamic Debug	libnndstlsd.so	
Static Release	libnndstlsz.a	
Static Debug	libnndstlszd.a	

7.1.2.5 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 7.8 Additional Libraries for Zero Copy Transfer Over Shared Memory](#).

Table 7.8 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Transfer Over Shared Memory Libraries ^[c]
Dynamic Release	libnndsmetp.so
Dynamic Debug	libnndsmetpd.so
Static Release	libnndsmetpz.a
Static Debug	libnndsmetpzd.a

7.1.3 How the Connex Libraries were Built

[Table 7.9 Library-Creation Details for QNX Architectures on the next page](#) shows the compiler flags that RTI used to build the *Connex* 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 QNX Platforms on page 62](#).

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

[b] OpenSSL libraries are in <NDDSHOME>/third_party/openssl-3.0.12/<architecture>/<format>/lib.

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

Table 7.9 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 -Werror-implicit-function-declaration
	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 -Werror-implicit-function-declaration
	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 -Werror-implicit-function-declaration
	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 -Werror-implicit-function-declaration
x64QNX7.1qcc_cxx8.3.0	Static Release	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DNDEBUG -DRTI_QNX -fPIC -Werror-implicit-function-declaration
	Static Debug	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64QNX7.1qcc_cxx8.3.0" -DRTI_QNX -fPIC -Werror-implicit-function-declaration
	Dynamic Release	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DNDEBUG -DRTI_QNX -fPIC -Werror-implicit-function-declaration
	Dynamic Debug	-Vgcc/8.3.0,gcc_ntox86_64 -Y_cxx -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64QNX7.1qcc_cxx8.3.0" -DRTI_QNX -fPIC -Werror-implicit-function-declaration
armv7QNX7.0.0qcc_cxx5.4.0 [a]	Static Release	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="\armv7QNX7.0.0qcc_cxx5.4.0" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O -DNDEBUG -Werror-implicit-function-declaration
	Static Debug	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="\armv7QNX7.0.0qcc_cxx5.4.0" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O0 -g -Werror-implicit-function-declaration
	Dynamic Release	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="\armv7QNX7.0.0qcc_cxx5.4.0" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O -DNDEBUG -fPIC -Werror-implicit-function-declaration
	Dynamic Debug	-Vgcc/5.4.0,gcc_ntoarmv7le -DCPU=ARMV7 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="\armv7QNX7.0.0qcc_cxx5.4.0" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O0 -g -fPIC -Werror-implicit-function-declaration

[a] Requires a hardware FPU in the processor and is compatible with systems that have hard-float libc.

Table 7.9 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv8QNX7.0.0qcc_cxx5.4.0	Static Release	-Vgcc/5.4.0.gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QNX7.0.0qcc_cxx5.4.0" -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O -DNDEBUG -Werror-implicit-function-declaration
	Static Debug	-Vgcc/5.4.0.gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QNX7.0.0qcc_cxx5.4.0" -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_cxx -O0 -g -Werror-implicit-function-declaration
	Dynamic Release	-Vgcc/5.4.0.gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QNX7.0.0qcc_cxx5.4.0" -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O -DNDEBUG -fPIC -Werror-implicit-function-declaration
	Dynamic Debug	-Vgcc/5.4.0.gcc_ntoaarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QNX7.0.0qcc_cxx5.4.0" -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_cxx -O0 -g -fPIC -Werror-implicit-function-declaration
x64QNX7.0.0qcc_gpp5.4.0	Static Release	-Vgcc/5.4.0.gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="x64QNX7.0.0qcc_gpp5.4.0" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O -DNDEBUG -Werror-implicit-function-declaration
	Static Debug	-Vgcc/5.4.0.gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="x64QNX7.0.0qcc_gpp5.4.0" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fPIC -fexceptions -v -Y_gpp -O0 -g -Werror-implicit-function-declaration
	Dynamic Release	-Vgcc/5.4.0.gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="x64QNX7.0.0qcc_gpp5.4.0" -D_GLIBCXX_USE_C99 -O -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O -DNDEBUG -fPIC -Werror-implicit-function-declaration
	Dynamic Debug	-Vgcc/5.4.0.gcc_ntox86_64 -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="x64QNX7.0.0qcc_gpp5.4.0" -D_GLIBCXX_USE_C99 -O0 -Wall -Wno-unknown-pragmas -fexceptions -v -Y_gpp -O0 -g -fPIC -Werror-implicit-function-declaration
armv8QNX7.1qcc_cxx8.3.0	Static Release	-Vgcc/8.3.0.gcc_ntoaarch64le -Y_gpp -fPIC -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET="armv8QNX7.1qcc_cxx8.3.0" -DNDEBUG -DRTI_QNX -Werror-implicit-function-declaration
	Static Debug	-Vgcc/8.3.0.gcc_ntoaarch64le -Y_gpp -fPIC -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET="armv8QNX7.1qcc_cxx8.3.0" -DRTI_QNX -Werror-implicit-function-declaration
	Dynamic Release	-Vgcc/8.3.0.gcc_ntoaarch64le -Y_gpp -fexceptions -DFD_SETSIZE=512 -O -DPtrIntType=long -DTARGET="armv8QNX7.1qcc_cxx8.3.0" -DNDEBUG -DRTI_QNX -fPIC -Werror-implicit-function-declaration
	Dynamic Debug	-Vgcc/8.3.0.gcc_ntoaarch64le -Y_gpp -fexceptions -DFD_SETSIZE=512 -O0 -g -DPtrIntType=long -DTARGET="armv8QNX7.1qcc_cxx8.3.0" -DRTI_QNX -fPIC -Werror-implicit-function-declaration

Table 7.9 Library-Creation Details for QNX Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
armv8QOS2.2qcc_cxx8.3.0	Static Release	-Vgcc/8.3.0.gcc_ntoarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QOS2.2qcc_cxx8.3.0" -O -fexceptions -Y_cxx -DNDEBUG -fPIC -Werror-implicit-function-declaration
	Static Debug	-Vgcc/8.3.0.gcc_ntoarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QOS2.2qcc_cxx8.3.0" -O0 -fexceptions -Y_cxx -g -fPIC -Werror-implicit-function-declaration
	Dynamic Release	-Vgcc/8.3.0.gcc_ntoarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QOS2.2qcc_cxx8.3.0" -O -fexceptions -Y_cxx -DNDEBUG -fPIC -Werror-implicit-function-declaration
	Dynamic Debug	-Vgcc/8.3.0.gcc_ntoarch64le -DFD_SETSIZE=512 -DPtrIntType=long -DTARGET="armv8QOS2.2qcc_cxx8.3.0" -O0 -fexceptions -Y_cxx -g -fPIC -Werror-implicit-function-declaration

7.2 Running Your Application

Table 7.10 [Running Instructions for QNX Architectures](#) provides details on the environment variables that must be set at run time for a QNX architecture.

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

Table 7.10 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]

7.3 Support for Modern C++ API

Connex provides two C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs. All supported QNX platforms support both.

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

[a] `${NDDSHOME}` represents the root directory of your *Connex* installation. `${LD_LIBRARY_PATH}` represents the value of the `LD_LIBRARY_PATH` variable prior to changing it to support *Connex*. 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 Connex Core Libraries User's Manual](#).

7.4 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.

7.5 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 *Connex* 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 *Connex* 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 *Connex* 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.

7.6 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

See also: [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#). These tables show which RTI products are supported for each platform.

7.7 Monotonic Clock Support

The monotonic clock (described in [Configuring the Clock per DomainParticipant, in the RTI Connex Core Libraries User's Manual](#)) is supported on all QNX platforms.

7.8 Thread Configuration

See [Table 7.11 Thread Settings for QNX Platforms](#) and [Table 7.12 Thread-Priority Definitions for QNX Platforms](#).

Table 7.11 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 7.11 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 7.12 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

7.8.1 Support for Controlling CPU Core Affinity for RTI Threads

Support for controlling CPU core affinity (described in *Controlling CPU Core Affinity* in the [RTI Connex Core Libraries User's Manual](#)) is available on all supported QNX platforms except arm-v8QOS2.2qcc_cxx8.3.0.

7.9 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on all QNX platforms except armv7QNX7.0.0qcc_cxx5.4.0. For information on using this script, see [2.5 Building with CMake on page 8](#).

7.10 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, in the RTI Connex Core Libraries User's Manual](#).

7.11 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) *Connex*t 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 *Connex*t 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 *Connex*t application that creates the mutex crashes—or terminates ungracefully—having only partially created the mutex. If this occurs, other *Connex*t 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 re-launching any application in the same DDS domain. The files to delete are:

- **`/dev/sem/RTIOSapiSharedMemoryMutex-<identifier>`**
- **`/dev/shmem/RTIOSapiSharedMemorySegment-<identifier>`**

Chapter 8 VxWorks Platforms

Table 8.1 Supported VxWorks Target Platforms in Connex 7.3.0 LTS lists the architectures supported on VxWorks® operating systems. You can build a VxWorks application by cross-compiling from your development host.

Table 8.1 Supported VxWorks Target Platforms in Connex 7.3.0 LTS

Operating System	CPU	Compiler	RTI Architecture Abbreviation
VxWorks 23.09	x64	llvm 16.0	x64Vx23.09Ivm16.0 x64Vx23.09Ivm16.0_rtp
The following rows show custom-supported target platforms, only available on demand. Please contact your RTI sales representative or sales@rti.com for more information on using these platforms:			
VxWorks 7.0 (SR0630)	x64	llvm 8.0.0.2	x64Vx7SR0630Ivm8.0.0.2 x64Vx7SR0630Ivm8.0.0.2_rtp (Custom support)
VxWorks 22.03	ppc	gcc 8.3.0	ppc32Vx22.03gcc8.3.0_rtp (Custom support)

8.1 Building Applications for VxWorks Platforms

The following notes apply to VxWorks 7-based platforms, including VxWorks 23.09.

- Compiling a *Connex* application for VxWorks depends on the development platform. For more information, such as specific compiler flags, see the *VxWorks Programmer's Guide*. [Table 8.7 Library-Creation Details for VxWorks Architectures on page 82](#) 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*.
- 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 *Connex* 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 *Connex* 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 *Connex* 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 + *Connex* libraries, and another static DKM containing the subscriber application + *Connex* libraries. When those two modules are loaded into the kernel, all the *Connex* 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 *Connex* libraries. With this approach, you can link this single DKM and still have the publisher and subscriber applications available in the kernel without duplication of symbols.

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

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

8.1.2 Required Libraries and Compiler Flags

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

[Table 8.2 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 *Connex*t features you want to use, you may need additional libraries; see [8.1.3 Additional Libraries for Other Features](#) on page 79.

Additional Documentation: See the [RTI Connex 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 8.2 Building Instructions for VxWorks Architectures

API	Library Format	Required RTI Libraries [a] [b]	Required Kernel Components	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnndscorez.a libnndscz.a libnndscppz.a or libnndscpp2z.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 -DRTI_64BIT
	Static Debug	libnndscorezd.a libnndsczd.a libnndscppzd.a or libnndscpp2zd.a librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnndscore.so libnndsc.so libnndscpp.so or libnndscpp2.so librticonnextmsgcpp.so or librticonnextmsgcpp2.so		
	Dynamic Debug	libnndscored.so libnndscd.so libnndscppd.so or libnndscpp2d.so librticonnextmsgcppd.so or librticonnextmsgcpp2d.so		

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

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

Table 8.2 Building Instructions for VxWorks Architectures

API	Library Format	Required RTI Libraries [a] [b]	Required Kernel Components	Required Compiler Flags
C	Static Release	libnddscorz.a libnddscz.a librticonnextmsgcz.a	INCLUDE_TIMESTAMP INCLUDE_POSIX_CLOCKS For RTI architectures with SMP support, also use: INCLUDE_TLS	-DRTI_VXWORKS -DRTI_CLANG -DRTI_64BIT
	Static Debug	libnddscorz.d.a libnddsczd.a librticonnextmsgcz.d.a		
	Dynamic Release	libnddscorz.so libnddsc.so librticonnextmsgc.so		
	Dynamic Debug	libnddscorz.d.so libnddscd.so librticonnextmsgcd.so		

8.1.3 Additional Libraries for Other Features

8.1.3.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported all VxWorks architectures. [Table 8.3 Additional Libraries for using RTI Distributed Logger](#) lists the additional libraries you will need in order to use *Distributed Logger*.

Table 8.3 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	librtidcd.so librtidlcppd.so

8.1.3.2 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 *Connex*t application is linked with the static release version of the *Connex*t 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.

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

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

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

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

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

Notes:

- Automatic loading of the dynamic monitoring library through QoS is not supported.
- Memory and CPU usage is not available in monitoring data.
- If you plan to use *static* libraries, the RTI library from [Table 8.4 Additional Libraries for Monitoring](#) must appear *first* in the list of libraries to be linked.

Table 8.4 Additional Libraries for Monitoring

Library Format	Monitoring Libraries ^[a]
Dynamic Release	librtmonitoring.so ^[b]
Dynamic Debug	librtmonitoringd.so ^[c]
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

8.1.3.3 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 8.5 Additional Libraries for Using Real-Time WAN Transport APIs](#). Select the file appropriate for your chosen library format.

For more information, see [Enabling Real-Time WAN Transport, in the RTI Connex 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 8.5 Additional Libraries for Using Real-Time WAN Transport APIs

Library Format	Real-Time WAN Transport Libraries ^[a]
Dynamic Release	libnndsrwt.so
Dynamic Debug	libnndsrwtd.so
Static Release	libnndsrwtz.a
Static Debug	libnndsrwtzd.a

8.1.3.4 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 8.6 Additional Libraries for Zero Copy Transfer Over Shared Memory](#).

Table 8.6 Additional Libraries for Zero Copy Transfer Over Shared Memory

Library Format	Zero Copy Over Shared Memory Libraries ^[b]
Dynamic Release	libnndsmetp.so
Dynamic Debug	libnndsmetpd.so
Static Release	libnndsmetpz.a
Static Debug	libnndsmetpzd.a

8.1.4 How the Connex Libraries were Built

[Table 8.2 Building Instructions for VxWorks Architectures on page 78](#) shows the compiler flags that RTI used to build the Connex 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 [8.1 Building Applications for VxWorks Platforms on page 75](#).

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

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

Table 8.7 Library-Creation Details for VxWorks Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
ppc32Vx22.03gcc8.3.0_rtp	Static Release	-DPtrIntType=long -DTARGET="ppc32Vx22.03gcc8.3.0_rtp" -DTOOL=gnu -DTOOL_FAMILY=gnu -D_USE_INIT_ARRAY -D_VX_CPU=VX_PPCE500MC -D__RTP__ -D_VXWORKS__ -D__ppc__ -D__ppc__ -D_vxworks -O -mabi=no-altivec -mno-altivec -mstrict-align -msecure-plt -fno-builtin -fno-strict-aliasing -DNDEBUG -Werror-implicit-function-declaration
	Static Debug	-DPtrIntType=long -DTARGET="ppc32Vx22.03gcc8.3.0_rtp" -DTOOL=gnu -DTOOL_FAMILY=gnu -D_USE_INIT_ARRAY -D_VX_CPU=VX_PPCE500MC -D__RTP__ -D_VXWORKS__ -D__ppc__ -D__ppc__ -D_vxworks -O0 -mabi=no-altivec -mno-altivec -mstrict-align -msecure-plt -fno-builtin -fno-strict-aliasing -g -Werror-implicit-function-declaration
x64Vx23.09llvm16.0	Static Release	--target=x86_64-wrs-vxworks -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 -O2 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=kernel -mno-implicit-float -mno-red-zone -fno-omit-frame-pointer -ftls-model=local-exec -DNDEBUG -std=c11 -Werror-implicit-function-declaration
	Static Debug	--target=x86_64-wrs-vxworks -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 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=kernel -mno-implicit-float -mno-red-zone -fno-omit-frame-pointer -ftls-model=local-exec -g -std=c11 -Werror-implicit-function-declaration
	Dynamic Release	--target=x86_64-wrs-vxworks -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 -O2 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=kernel -mno-implicit-float -mno-red-zone -fno-omit-frame-pointer -ftls-model=local-exec -DNDEBUG -std=c11 -Werror-implicit-function-declaration
	Dynamic Debug	--target=x86_64-wrs-vxworks -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 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=kernel -mno-implicit-float -mno-red-zone -fno-omit-frame-pointer -ftls-model=local-exec -g -std=c11 -Werror-implicit-function-declaration
x64Vx23.09llvm16.0_rtp	Static Release	--target=x86_64-wrs-vxworks -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -D_VX_TOOL=llvm -D_VX_TOOL_FAMILY=llvm -D__ELF__ -D__RTP__ -D_VXWORKS__ -D_vxworks -O2 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=small -fasn -fno-omit-frame-pointer -DNDEBUG -std=c11 -Werror-implicit-function-declaration
	Static Debug	--target=x86_64-wrs-vxworks -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -D_VX_TOOL=llvm -D_VX_TOOL_FAMILY=llvm -D__ELF__ -D__RTP__ -D_VXWORKS__ -D_vxworks -O0 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=small -fasn -fno-omit-frame-pointer -O0 -g -std=c11 -Werror-implicit-function-declaration
	Dynamic Release	--target=x86_64-wrs-vxworks -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -D_VX_TOOL=llvm -D_VX_TOOL_FAMILY=llvm -D__ELF__ -D__RTP__ -D_VXWORKS__ -D_vxworks -O2 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=small -fasn -fno-omit-frame-pointer -DNDEBUG -fPIC -std=c11 -Werror-implicit-function-declaration
	Dynamic Debug	--target=x86_64-wrs-vxworks -D_HAVE_TOOL_XTORS -D_USE_INIT_ARRAY -D_VX_CPU=_VX_CORE -D_VX_TOOL=llvm -D_VX_TOOL_FAMILY=llvm -D__ELF__ -D__RTP__ -D_VXWORKS__ -D_vxworks -O0 -m64 -march=core2 -fno-builtin -fno-strict-aliasing -nostdlibinc -mcmmodel=small -fasn -fno-omit-frame-pointer -g -fPIC -std=c11 -Werror-implicit-function-declaration

Table 8.7 Library-Creation Details for VxWorks Architectures

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

8.2 Running User Applications

Table 8.8 Running Instructions for VxWorks Architectures below provides details on the environment variables that must be set at runtime for a VxWorks architecture.

Table 8.8 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_connex_libs>;<path_to_libc>" [a]
	Static	None required

8.3 Known Defects

Defect V7COR-8916 can cause unpredictable segmentation faults in VxWorks 23.09 RTP applications. There is a patch available from Wind River, which they will provide upon request through their Support Network. This patch is required in order to support the x64Vx23.09llvm16.0 rtp architecture.

8.4 Increasing the Stack Size

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

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

8.5 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] To run dynamic RTP executables, the runtime `libc.so` library must be accessible. See the VxWorks Application Programmer's guide for more information.

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

The *Connex* Professional, Research, and LM packages include support for the Request-Reply Communication Pattern, for all platforms in [Table 8.1 Supported VxWorks Target Platforms in Connex 7.3.0 LTS on page 75](#) 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 *Connex* DKMs (**libnndscore.so**, **libnndsc.so**, **libnndscpp.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 *Connex* libraries inside the DKM (such as **libnndscorez.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 *Connex* will be loaded once per DKM.
- Perform a partial linkage: This involves building your application without linking against the *Connex* libraries. Later, at load time, you will need to load into the kernel the required *Connex* libraries and your application DKM. This is recommended if you plan to have more than one DKM using *Connex*.

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

8.7 Requirement for Restarting Applications

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

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

All *Connex* 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.

8.8 Support for Modern C++ API and Remote Procedure Calls (RPCs)

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

- The Traditional C++ API supports C++98 compilers or newer. It is supported on all VxWorks Platforms.
- The Modern C++ API requires C++11 compilers or newer. It is supported as noted in [Table 8.9](#)

Table 8.9 C++ API and RPC Support

Operating System	RTI Architecture	Modern C++ API and RPC Support
VxWorks 23.09	x64Vx23.09Ivm16.0 x64Vx23.09Ivm16.0_rtp	Yes (without RPC support)
VxWorks 7.0 (SR0630)	x64Vx7SR0630Ivm8.0.0.2	No
	x64Vx7SR0630Ivm8.0.0.2_rtp	Yes (without RPC support)
VxWorks 22.03	ppc32Vx22.03gcc8.3.0_rtp	Yes (without RPC support)

For more information, see [Traditional vs. Modern C++, in the RTI Connex Core Libraries User's Manual](#).

RPC is an experimental feature, only available for the Modern C++ API on the architectures noted in [Table 8.9](#).

See [Remote Procedure Calls, in the RTI Connex Core Libraries User's Manual](#).

8.9 Multicast Support

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

8.10 Transports

- **Shared memory:** Shared memory is supported and enabled by default on all VxWorks architectures. See also:

- [8.10.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID below](#)
- [8.10.2 How To Run Connex Libraries in Kernels Built without Shared Memory below](#)
- **UDPv4:** Supported and enabled by default.
- **UDPv6:** Supported. No Traffic Class support.
- **TCP/IPv4:** Not supported.

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

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

8.10.2 How To Run Connex Libraries in Kernels Built without Shared Memory

Since *Connex* 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 *Connex* libraries are loaded after boot, or linked at kernel build time.

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

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

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

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

8.11 Unsupported Features

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
- TCP v4 transport

See also: [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#). These tables show which RTI products are supported for each platform.

8.12 Monotonic Clock Support

The monotonic clock (described in [Configuring the Clock per DomainParticipant, in the RTI Connex Core Libraries User's Manual](#)) is supported on all VxWorks platforms.

8.13 Use of Real-Time Clock

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

8.14 Thread Configuration

See these tables:

- [Table 8.10 Thread Setting for VxWorks Platforms on the next page](#)
- [Table 8.11 Thread-Priority Definitions for VxWorks Platforms on the next page](#)
- [Table 8.12 Thread Kinds for VxWorks Platforms on page 90](#)

Table 8.10 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
Event thread	mask	DDS_THREAD_SETTINGS_STDIO DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	110
	stack_size	4 * 30 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	71
	stack_size	4 * 30 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 8.11 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

Table 8.11 Thread-Priority Definitions for VxWorks Platforms

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_NORMAL	100
THREAD_PRIORITY_BELOW_NORMAL	110
THREAD_PRIORITY_LOW	120

Table 8.12 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 9 Windows Platforms

This release supports the Windows platforms in [Table 9.1](#) .

Table 9.1 Supported Windows Platforms in Connex 7.3.0 LTS

CPU	Visual Studio® Version	RTI Architecture	Supported Target OS	Minimum MSVC++ Redistributable	.NET Version	JDK Version
x64	VS 2017 Update 2, VS 2019, VS 2022	x64Win64VS2017	Windows 10, 11 Windows Server 2012 R2 - 2022	Microsoft Visual C++ Redistributable for Visual Studio 2017 for x64	.NET Standard 2.0	AdoptOpenJDK 17.0.6
Arm v8	VS 2022	arm64Win64VS2022	Windows 11	Microsoft Visual C++ Redistributable for Visual Studio 2022 for Arm v8		
The following rows show custom-supported target platforms, only available on demand. Please contact your RTI sales representative or sales@rti.com for more information on using these platforms:						
x64	VS 2015 Update 3	x64Win64VS2015 <i>(Custom support)</i>	Windows 10 Windows Server 2012 R2 - 2022	Microsoft Visual C++ 2015 Redistributable Update 3 for x64	.NET Standard 2.0	AdoptOpenJDK 17.0.6
x86	VS 2015 Update 3	i86Win32VS2015 <i>(Custom support)</i>	Windows 10	Microsoft Visual C++ 2015 Redistributable Update 3 for x86		
	VS 2017 Update 2, VS 2019	i86Win32VS2017 <i>(Custom support)</i>	Windows Server 2016	Microsoft Visual C++ 2017 Redistributable Update 3 for x86		

For a list of Windows OS versions supported by your Visual Studio version, please see official Microsoft documentation.

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 [Connex C# API Reference](#).

9.1 Building Applications for Windows Platforms

Before starting to build your *Connex* application, make sure that:

- A supported version of Visual Studio is installed on the machine you are using to build your application (see [Table 9.1](#) above), and you have installed a valid SDK for your target platform.
- You have installed *Connex* and an RTI architecture package that supports your desired target platform on the build machine. You can find more instructions in the [RTI Connex Getting Started Guide](#).

Note for Windows on Arm architectures: There is no Arm64 host installer. If you want to link your application against the dynamic version of the *Connex* libraries, you will need to install the x64 host and the target package on the target machine, or manually copy to the target the *Connex* installation folder with the desired libraries installed.

- You have set up your build environment following the basic instructions in [Chapter 2 Building Applications—Notes for All Platforms on page 5](#)

9.2 Configuring the Build of Your Connex Application

To compile a *Hello World* application with *Connex*, we recommend starting with one of the VS Project examples provided with *Connex* or generating an example using *RTI Code Generator (rtiddsgen)*.

If you want to create your own project files from scratch, or build your application without using a project file, follow these steps:

1. Set up your build environment following the basic instructions in [Chapter 2 Building Applications—Notes for All Platforms on page 5](#)
2. Add the path to your *Connex* installation folder to a environment variable called NDDSHOME:

```
set NDDSHOME=<Path to your connex installation folder>
```

3. Include these additional include directories (/I)
 - \$(NDDSHOME)\include
 - \$(NDDSHOME)\include\ndds
4. Include the following path in your PATH environment variable or as an additional Libpath (/LIBPATH):
 - \$(NDDSHOME)\lib<architecture>
5. Link against the Windows C Run-Time Libraries. All *Connex* libraries must link against the dynamic Windows C Run-Time (CRT).

- a. Specify the runtime library to use, based on the *Connex* library version you want to link against:
 - If you are using a Release version: Multithread-specific and DLL-specific (/MD)
 - If you are using a Debug version: Multithread-specific and DLL-specific debug (/MDd)
 - b. Remove the following default libraries from the list of libraries to be searched for when resolving external references: (/NODEFAULTLIB)
 - libcmtd
 - libcmtd
6. If you want to use the MFC library in you application, you must link against the *dynamic* version. (If you use the static version, your *Connex* application may stop receiving DDS samples once the Windows sockets are initialized.)
 7. Specify that the linker should use the required RTI and system libraries, and the compiler flags from [Table 9.2 Building Instructions for Windows Architectures below](#), depending on the version of the libraries you plan to link against. Make sure you are consistent in your use of static (.lib), dynamic (.dll), debug and release versions of the libraries. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

Table 9.2 Building Instructions for Windows Architectures

API	Library Format	RTI Libraries or Jar Files ^[a] ^[b]	Required System Libraries	Required Compiler Flags
C	Static Release	nddscorez.lib nddscz.lib rticonnextmsgcz.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	WIN32_LEAN_AND_MEAN WIN32 If linking against <i>dynamic</i> RTI libraries, add NDDS_DLL_VARIABLE If linking against a <i>static</i> RTI libraries and using the <i>RTI Security Plugins</i> , add RTI_STATIC
	Static Debug	nddscorezd.lib nddsczd.lib rticonnextmsgczd.lib		
	Dynamic Release	nddscore.lib nddsc.lib rticonnextmsgc.lib		
	Dynamic Debug	nddscored.lib nddscd.lib rticonnextmsgcd.lib		

[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 9.2 Building Instructions for Windows Architectures

API	Library Format	RTI Libraries or Jar Files ^[a] ^[b]	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	nddscorez.lib nddscz.lib nddscppz.lib or nddscpp2z.lib rticonnextmsgcppz.lib or rticonnextmsgcpp2z.lib	netapi32.lib advapi32.lib user32.lib ws2_32.lib	WIN32_LEAN_AND_MEAN WIN32 If linking against <i>dynamic</i> RTI libraries, add NDDS_DLL_VARIABLE If linking against a <i>static</i> RTI libraries and using the <i>RTI Security Plugins</i> , add RTI_STATIC
	Static Debug	nddscorezd.lib nddsczd.lib nddscppzd.lib or nddscpp2zd.lib rticonnextmsgcppzd.lib or rticonnextmsgcpp2zd.lib		
	Dynamic Release	nddscore.lib nddsc.lib nddscpp.lib or nddscpp2.lib rticonnextmsgcpp.lib or rticonnextmsgcpp2.lib		
	Dynamic Debug	nddscored.lib nddscd.lib nddscppd.lib or nddscpp2d.lib rticonnextmsgcppd.lib or rticonnextmsgcpp2d.lib		
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.

9.2.1 Additional Libraries for Other Features

9.2.1.1 Libraries Required for Distributed Logger

RTI Distributed Logger is supported on all Windows platforms. [Table 9.3 Additional Libraries for using RTI Distributed Logger](#) lists the additional libraries you will need to use *Distributed Logger*.

Table 9.3 Additional Libraries for using RTI Distributed Logger

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

9.2.1.2 Libraries Required for Monitoring

To use the Monitoring APIs, reference the libraries in [Table 9.4 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 *Connex* application is linked with the static release version of the *Connex* 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, causing the process to terminate.

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

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

Table 9.4 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

9.2.1.3 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 [RTI Real-Time WAN Transport Installation Guide](#) for details.

Using *Real-Time WAN Transport* requires one of the libraries in [9.2.1 Additional Libraries for Other Features](#). Select the file appropriate for your chosen library format.

For more information, see [Enabling Real-Time WAN Transport, in the RTI Connex Core Libraries User's Manual](#).

Table 9.5 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
Static Release	nddsrwtz.lib
Static Debug	nddsrwtzd.lib

For details on the OpenSSL libraries, see [9.2.3 Location of OpenSSL Libraries on page 100](#).

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

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

9.2.1.4 Libraries Required for RTI TCP Transport

To use the TCP Transport APIs, reference the libraries in [Table 9.6 Additional Libraries for Using RTI TCP Transport APIs](#).

Table 9.6 Additional Libraries for Using RTI TCP Transport APIs

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

If you are also using *RTI TLS Support*, see [Table 9.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled](#). (Select the files appropriate for your chosen library format.)

Table 9.7 Additional Libraries for using RTI TCP Transport APIs with TLS Enabled

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

For details on the OpenSSL libraries, see [9.2.3 Location of OpenSSL Libraries on page 100](#).

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

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

9.2.1.5 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, reference the libraries in [Table 9.8 Additional Libraries for Zero Copy Transfer Over Shared Memory](#).

Table 9.8 Additional Libraries for Zero Copy Transfer Over Shared Memory

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

9.2.2 How the Connex Libraries were Built

[Table 9.9 Library-Creation Details for Windows Architectures](#) shows the compiler flags that RTI used to build the Connex 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 [9.1 Building Applications for Windows Platforms on page 92](#).

Table 9.9 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
arm64Win64VS2022	Static Release	-D_WIN32_WINNT=0x0600 -D_WINDOWS -D_CRT_SECURE_NO_DEPRECATED -DWIN32 -DWIN32_LEAN_AND_MEAN -DCOMPILED_FROM_DSP -DSTDC99 /W3 /Oy- /MD /O2 /Ob0 /DNDEBUG /nodefaultlib:libc.lib /nodefaultlib:libcd.lib /defaultlib:Winmm.lib /we4013
	Static Debug	-D_WIN32_WINNT=0x0600 -D_WINDOWS -D_CRT_SECURE_NO_DEPRECATED -DWIN32 -DWIN32_LEAN_AND_MEAN -DCOMPILED_FROM_DSP -DSTDC99 /W3 /Oy- /MDd /Zi /Ob0 /Od /RTC1 /nodefaultlib:libc.lib /nodefaultlib:libcd.lib /defaultlib:Winmm.lib /we4013
	Dynamic Release	-D_WIN32_WINNT=0x0600 -D_WINDOWS -D_CRT_SECURE_NO_DEPRECATED -DWIN32 -DWIN32_LEAN_AND_MEAN -DCOMPILED_FROM_DSP -DSTDC99 /W3 /Oy- /MD /O2 /Ob0 /DNDEBUG /we4013
	Dynamic Debug	-D_WIN32_WINNT=0x0600 -D_WINDOWS -D_CRT_SECURE_NO_DEPRECATED -DWIN32 -DWIN32_LEAN_AND_MEAN -DCOMPILED_FROM_DSP -DSTDC99 /W3 /Oy- /MDd /Zi /Ob0 /Od /RTC1 /we4013

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

Table 9.9 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
i86Win32VS2015	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /ZI /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Static Debug	/W3 -DSTDC99 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
	Dynamic Release	/W3 -DSTDC99 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /ZI /MD /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Dynamic Debug	/W3 -DSTDC99 -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
i86Win32VS2017	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /ZI /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /Oy- /ZI /MD /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=i80586 -DTARGET="\x86Win32VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
x64Win64VS2015	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /ZI /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /ZI /MD /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2015" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013

Table 9.9 Library-Creation Details for Windows Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Win64VS2017	Static Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Static Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /nodefaultlib:"libcmt.lib" /defaultlib:"msvcrt.lib" /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
	Dynamic Release	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /O2 /Zi /MD /EHsc -D_CRT_SECURE_NO_DEPRECATED -DNDEBUG -c /we4013
	Dynamic Debug	/W3 -DSTDC99 /FS -DPtrIntType=long -DCSREAL_IS_FLOAT -DCPU=AMD64 -DTARGET="\x64Win64VS2017" -DWIN32 -D_WINDOWS -D_WIN32_WINNT=0x0600 -DWIN32_LEAN_AND_MEAN /Od /ZI /MDd /EHsc /RTC1 -D_CRT_SECURE_NO_DEPRECATED -c /we4013
Windows architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

9.2.3 Location of OpenSSL Libraries

The OpenSSL libraries are installed here:

- OpenSSL **.lib** files are in:
`<NDDSHOME>\third_party\openssl-3.0.12\<architecture>\<format>\lib.`
- OpenSSL **.dll** files are in:
`<NDDSHOME>\third_party\openssl-3.0.12\<architecture>\<format>\bin.`

Where:

- `<architecture>` is your architecture string, as listed in [Table 9.1 Supported Windows Platforms in Connex 7.3.0 LTS on page 91](#), such as `x64Win64VS2017`.
- `<format>` is `debug`, `release`, `static_debug`, or `static_release`.

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

9.3 Running Your Applications

Before running a *Connex* application, make sure that:

1. You have a valid Visual Studio Redistributable installed on the target machine; which redistributable depends on your target architecture, see [Table 9.1 Supported Windows Platforms in Connex 7.3.0 LTS on page 91](#).
2. The location of the RTI libraries for your target architecture are in the PATH environment variable as noted in [Table 9.10 Running Instructions for Windows Architectures below](#).

In some cases, when components need third-party libraries, you may need to add other locations, and set up other variables. We recommend using the provided `rtisetenv <architecture>` script to set up your environment. (See [Set Up Environment Variables, in the RTI Connex Getting Started Guide](#).)

For a detailed explanation on how to run your applications, see [Run the Applications, in the RTI Connex Getting Started Guide](#).

Table 9.10 Running Instructions for Windows Architectures

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

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 or manually copy the files to your target.

9.4 Support for the Modern C++ API

Connex provides two C++ APIs, which we refer to as the "Traditional C++" and "Modern C++" APIs. Both are supported on all Windows platforms.

- 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 Connex Core Libraries User's Manual](#).

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

9.5 Support for the .NET (C#) API

The C# API is supported on Intel x64 and Arm v8 CPUs.^[a] For more information on .NET, see the [Connex C# API Reference](#).

9.6 Support for the Python API

The Python API is supported for Python 3.6 - 3.12, on Intel x64 CPUs. For more information, see the [Connex Python API Reference](#).

9.7 Multicast Support

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

9.8 Transports

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

9.9 Unsupported Features

These features are not supported on Windows platforms:

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

These features are not supported on Windows platforms with Arm v8 CPUs:

- Python API
- `rtiddsgen` server
- *RTI Connector for JavaScript*

[a] The C# API is also supported on Intel x86 with Visual Studio 2017 (i86Win32VS2017, a custom-supported platform).

See also: [Supported Platforms tables in the RTI Connex Core Libraries Release Notes](#). These tables show which RTI products are supported for each platform.

9.10 Monotonic Clock Support

The monotonic clock (described in [Configuring the Clock per DomainParticipant, in the RTI Connex Core Libraries User's Manual](#)) is supported on all Windows platforms.

9.11 Thread Configuration

See these tables:

- [Table 9.11 Thread Settings for Windows Platforms](#)
- [Table 9.12 Thread-Priority Definitions for Windows Platforms](#)
- [Table 9.13 Thread Kinds for Windows Platforms](#)

Table 9.11 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

Table 9.11 Thread Settings for Windows Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	2
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 9.12 Thread-Priority Definitions for Windows Platforms

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

Table 9.13 Thread Kinds for Windows Platforms

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

9.12 Support for 'Find Package' CMake Script

The 'Find Package' CMake script is supported on all Windows platforms except those on x86 CPUs.

For information on using this script, see [2.5 Building with CMake on page 8](#).

[a] See Windows manuals for additional information.

9.13 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.

9.14 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.

On 32-bit Windows architectures: You must 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 Connex Core Libraries User's Manual](#).

9.15 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, in the RTI Connex Core Libraries User's Manual](#).

9.16 Domain ID Support

On Windows platforms, you should avoid using ports 49152 through 65535 for inbound traffic. *Connex*'s ephemeral ports (see [Ports Used for Communication, in the RTI Connex 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](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.
```