

# **RTI Connex Core Libraries**

**Platform Notes**

**Version 7.6.0**



## **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 agreed to in writing by 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](https://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**

### **Early Access Software**

“Real-Time Innovations, Inc. (“RTI”) licenses this Early Access release software (“Software”) to you subject to your agreement to all of the following conditions:

- (1) you may reproduce and execute the Software only for your internal business purposes, solely with other RTI software licensed to you by RTI under applicable agreements by and between you and RTI, and solely in a non-production environment;
- (2) you acknowledge that the Software has not gone through all of RTI’s standard commercial testing, and is not maintained by RTI’s support team;
- (3) the Software is provided to you on an “AS IS” basis, and RTI disclaims, to the maximum extent permitted by applicable law, all express and implied representations, warranties and guarantees, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, satisfactory quality, and non-infringement of third party rights;

(4) any such suggestions or ideas you provide regarding the Software (collectively , “Feedback”), may be used and exploited in any and every way by RTI (including without limitation, by granting sub-licenses), on a non-exclusive, perpetual, irrevocable, transferable, and worldwide basis, without any compensation, without any obligation to report on such use, and without any other restriction or obligation to you; and

(5) TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL RTI BE LIABLE TO YOU FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, EXEMPLARY OR PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR FOR LOST PROFITS, LOST DATA, LOST REPUTATION, OR COST OF COVER, REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING WITHOUT LIMITATION, NEGLIGENCE), STRICT PRODUCT LIABILITY OR OTHERWISE, WHETHER ARISING OUT OF OR RELATING TO THE USE OR INABILITY TO USE THE SOFTWARE, EVEN IF RTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.”

### *Deprecations and Removals*

*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.

Any deprecations or removals noted in RTI's documentation 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. RTI's current standard terms and support and maintenance policies are available at <https://www.rti.com/terms>.

### **Technical Support**

Real-Time Innovations, Inc.

232 E. Java Drive

Sunnyvale, CA 94089

Phone: (408) 990-7444

Email: [support@rti.com](mailto:support@rti.com)

Website: <https://support.rti.com/>

# Contents

---

## Chapter 1 Introduction

1.1 Paths Mentioned in Documentation .....	2
--	---

## Chapter 2 Building Applications—Notes for All Platforms

2.1 Running on a Computer Not Connected to a Network .....	5
2.2 Connex Header Files — All Platforms .....	5
2.3 Choosing the Right Libraries .....	6
2.3.1 Required Libraries .....	6
2.3.2 Mixing Static and Dynamic Libraries is not Supported .....	7
2.4 Building for Java Platforms .....	7
2.5 Building with CMake .....	8
2.6 Building Generated Code .....	8

## Chapter 3 Android Platforms

3.1 Summary of Android Platforms and Supported Features .....	9
3.1.1 Monotonic Clock Support .....	12
3.2 Building Applications for Android Platforms .....	12
3.2.1 Required Libraries and Compiler Flags .....	12
3.2.2 Additional Libraries for Other Features .....	16
3.2.3 Target Configuration .....	18
3.2.4 ‘Release’ and ‘Debug’ Terminology .....	18
3.2.5 How the Connex Libraries were Built .....	19
3.3 Running Your Applications .....	19
3.4 Unsupported Features .....	20
3.5 Thread Configuration .....	20
3.5.1 Thread Settings and Thread-Priority Definitions .....	20
3.5.2 Automatic Thread-Specific Storage Cleanup .....	21
3.6 Socket Buffer Size Configuration .....	21

---

3.7 Third-Party Software Versions Used for Android 12 Development and Testing .....	22
<b>Chapter 4 INTEGRITY Platforms</b>	
4.1 Building Applications for INTEGRITY Platforms .....	24
4.1.1 Required Libraries and Compiler Flags .....	24
4.1.2 Required Patch for INTEGRITY 11.0.4 Platforms .....	26
4.1.3 Additional Libraries for Other Features .....	26
4.2 Running User Applications .....	28
4.3 Thread Configuration .....	28
4.3.1 Thread Settings and Thread-Priority Definitions .....	28
4.3.2 Socket-Enabled and POSIX-Enabled Threads are Required .....	29
4.3.3 Automatic Thread-Specific Storage Cleanup .....	30
4.4 Socket Buffer Size Configuration .....	30
4.5 Running over IP Backplane on a Dy4 Champ-AVII Board .....	30
4.6 Out-of-the-box Transport Compatibility with Other ConnexT Platforms .....	31
4.6.1 Smaller Shared-Memory Receive-Resource Queue Size .....	31
4.6.2 Using Shared Memory on INTEGRITY Systems .....	32
4.6.3 Shared Memory Limitations on INTEGRITY Systems .....	33
4.7 Using rtiddsping and rtiddsspy on INTEGRITY Systems .....	33
4.8 Issues with INTEGRITY Systems .....	33
4.8.1 Delay When Writing to Unreachable Peers .....	33
4.8.2 Linking with 'libivfs.a' without a File System .....	34
4.8.3 Compiler Warnings Regarding Unrecognized #pragma Directives .....	34
<b>Chapter 5 Linux Platforms</b>	
5.1 Summary of Linux Platforms and Supported Features .....	35
5.1.1 Monotonic Clock Support .....	40
5.1.2 Support for 'Find Package' CMake Script .....	40
5.1.3 Backtrace Support .....	40
5.1.4 Limitations of FACE Architectures .....	40
5.2 Building Applications for Linux Platforms .....	41
5.2.1 Required Libraries and Compiler Flags .....	41
5.2.2 Additional Libraries for Other Features .....	43
5.2.3 Linux Compatibility and Determining Factors .....	47
5.2.4 How the ConnexT Libraries were Built .....	49
5.3 Running Your Applications .....	50
5.3.1 Shared Memory Support .....	51
5.4 Thread Configuration .....	51

---

5.4.1 Thread Settings, Thread-Priority Definitions, and Thread Kinds .....	51
5.4.2 Using REALTIME_PRIORITY .....	53
5.4.3 Automatic Thread-Specific Storage Cleanup .....	53
5.5 Socket Buffer Size Configuration .....	53
<b>Chapter 6 macOS Platforms</b>	
6.1 Summary of macOS Platforms and Supported Features .....	55
6.1.1 Support for 'Find Package' CMake Script .....	58
6.1.2 Backtrace Support .....	58
6.2 Building Applications for macOS Platforms .....	58
6.2.1 Additional Libraries for Other Features .....	60
6.2.2 How the Connex Libraries were Built .....	63
6.3 Running User Applications .....	65
6.4 System Integrity Protection (SIP) .....	65
6.4.1 SIP and Java Applications .....	65
6.4.2 SIP and Connex Tools, Infrastructure Services, and Utilities .....	66
6.5 Thread Configuration .....	67
6.5.1 Thread Settings and Thread-Priority Definitions .....	67
6.5.2 Automatic Thread-Specific Storage Cleanup .....	68
6.6 Socket Buffer Size Configuration .....	68
6.7 Resolving NDDUtility_sleep() Issues .....	69
<b>Chapter 7 QNX Platforms</b>	
7.1 Summary of QNX Platforms and Supported Features .....	70
7.1.1 Monotonic Clock Support .....	74
7.1.2 Support for 'Find Package' CMake Script .....	74
7.1.3 Notes about Transports .....	74
7.2 Building Applications for QNX Platforms .....	74
7.2.1 Required Change for Building with C++ Libraries .....	77
7.2.2 Additional Libraries for Other Features .....	77
7.2.3 How the Connex Libraries were Built .....	80
7.3 Running Your Application .....	81
7.4 Thread Configuration .....	82
7.4.1 Thread Settings and Thread-Priority Definitions .....	82
7.4.2 Support for Controlling CPU Core Affinity for RTI Threads .....	83
7.4.3 Automatic Thread-Specific Storage Cleanup .....	83
7.5 Socket Buffer Size Configuration .....	83
7.6 Restarting Applications on QNX Systems .....	84

---

---

## Chapter 8 VxWorks Platforms

8.1 Building Applications for VxWorks Platforms .....	85
8.1.1 Libraries for RTP Mode on VxWorks Systems .....	86
8.1.2 Required Libraries and Compiler Flags .....	86
8.1.3 Additional Libraries for Other Features .....	88
8.2 Running User Applications .....	91
8.3 Known Defects .....	91
8.4 Increasing the Stack Size .....	91
8.5 Enabling Floating Point Coprocessor in Kernel Tasks .....	92
8.6 Downloadable Kernel Modules (DKM) for Kernel Mode on VxWorks Systems .....	92
8.7 Requirement for Restarting Applications .....	92
8.8 Transports .....	93
8.8.1 Shared-Memory Communication between Applications Running in Kernel Mode and RTP Requires Explicitly Set Participant ID .....	93
8.8.2 How To Run Connex Libraries in Kernels Built without Shared Memory .....	93
8.9 Unsupported Features .....	94
8.10 Monotonic Clock Support .....	94
8.11 Use of Real-Time Clock .....	94
8.12 Thread Configuration .....	94
8.12.1 Thread Settings, Thread-Priority Definitions, and Thread Kinds .....	94
8.12.2 Automatic Thread-Specific Storage Cleanup .....	96
8.13 Socket Buffer Size Configuration .....	96

## Chapter 9 Windows Platforms

9.1 Summary of Windows Platforms and Supported Features .....	98
9.1.1 Monotonic Clock Support .....	101
9.1.2 Support for 'Find Package' CMake Script .....	101
9.1.3 Backtrace Support .....	101
9.2 Building Applications for Windows Platforms .....	102
9.3 Configuring the Build of Your Connex Application .....	102
9.3.1 Additional Libraries for Other Features .....	105
9.3.2 How the Connex Libraries were Built .....	109
9.3.3 Location of OpenSSL Libraries .....	110
9.4 Running Your Applications .....	110
9.5 Thread Configuration .....	111
9.5.1 Thread Settings, Thread-Priority Definitions, and Thread Kinds .....	111
9.5.2 Automatic Thread-Specific Storage Cleanup .....	113
9.6 Socket Buffer Size Configuration .....	113

---

---

9.7 Domain ID Support .....	114
9.8 Requirements when Using Microsoft Visual Studio Code .....	114
9.8.1 When Using Visual Studio 2017 — Redistributable Package Requirement .....	115
9.8.2 When Using Visual Studio 2019 — Redistributable Package Requirement .....	115
9.8.3 When Using Visual Studio 2022 — Redistributable Package Requirement .....	115

# Chapter 1 Introduction

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

A *platform* refers to the combination of your target machine's OS version, CPU, and toolchain (compiler or Visual Studio). Each platform has an RTI architecture name, which is a shorthand way to identify the platform. The "target" is the machine where you will deploy your completed application. (As opposed to a "host", which is where you will be developing the application.)

For example, if you have a 64-bit Windows machine with Visual Studio® 2017, the architecture name is **x64Win64VS2017**. For a 64-bit Linux machine with gcc version 8.5.0, the architecture name is **armv8Linux4gcc8.5.0**.

**Not all platforms are supported in every release.** Please refer to the *Supported Platforms* section of the [RTI Connex Core Libraries Release Notes](#) to see which platforms are supported in this release. Custom-supported target platforms are available on demand. Contact [sales@rti.com](mailto:sales@rti.com).

For each supported OS, this document describes:

- 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

- Thread configuration
- Other platform-specific information

## 1.1 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

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

- macOS® systems:  
***/Applications/rti\_connex\_dds-7.6.0***
- Linux systems, non-*root* user:  
***/home/<your user name>/rti\_connex\_dds-7.6.0***
- Linux systems, *root* user:  
***/opt/rti\_connex\_dds-7.6.0***
- Windows® systems, user without Administrator privileges:  
***<your home directory>\rti\_connex\_dds-7.6.0***
- Windows systems, user with Administrator privileges:  
***C:\Program Files\rti\_connex\_dds-7.6.0***

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_connex_dds-7.6.0\bin\rtiddsgen"
```

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

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

- **API Reference HTML documentation**

Default path: **<NDDSHOME>/doc/api/connex\_dds/api\_<language>**

- **<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/7.6.0/examples***
- Linux systems: ***/home/<your user name>/rti\_workspace/7.6.0/examples***
- Windows systems: ***<your Windows documents folder>\rti\_workspace\7.6.0\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 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 35](#)
  - [Chapter 9 Windows Platforms on page 98](#)
- 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*, you must specify:

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

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

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

## 2.2 Connex Header Files – All Platforms

You must include the appropriate *Connex* 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 API	Header Files
C	<code>#include "ndds/ndds_c.h"</code>
C++	<code>#include "ndds/ndds_cpp.h"</code>
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*. See [1.1 Paths Mentioned in Documentation on page 2](#).

**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 included 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, dynamic 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.

#### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*, *debug* libraries are created with debug symbols to facilitate debugging with **gdb**, for example. *Release* libraries do not contain debug information.

The *Connex* debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production

environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

You must choose *either* release or debug libraries. Mixing release and debug libraries is not supported.

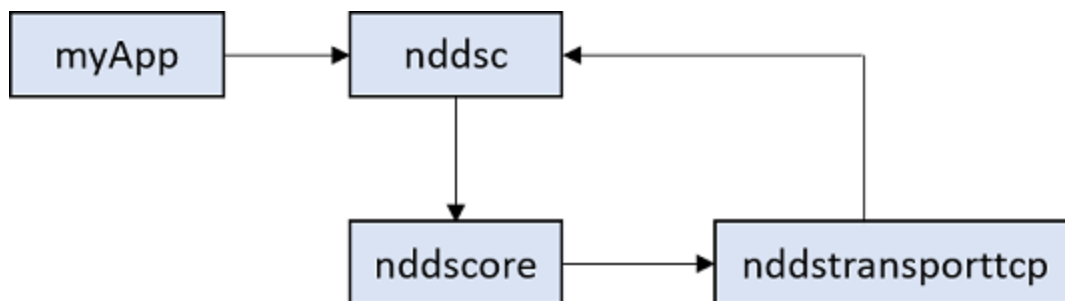
## 2.3.2 Mixing Static and Dynamic Libraries is not Supported

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.

## 2.6 Building Generated Code

For detailed information on how to build generated code as part of a *Connex* application, please refer to [Building Generated Code in the RTI Code Generator User's Manual](#).

# Chapter 3 Android Platforms

## 3.1 Summary of Android Platforms and Supported Features

The following table shows the supported products/features for each architecture. A ● means supported in this release, an empty cell means unsupported in this release.

**Table 3.1 Android Systems - Supported Features**

RTI Architecture <sup>[a]</sup>	Supported Features						
Languages							
	Java <sup>[b]</sup>	Python <sup>[c]</sup>	.NET (C#) <sup>[d]</sup>	C/C++	Modern C++ <sup>[e], [f]</sup>		
Android 12, 14 <sup>[g]</sup> arm64Android12clang12.0.8ndkr23b	●			●	●		
Android AOSP 14 <sup>[h]</sup> arm64AndroidAOSP14clang17.0.2				●	●		

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Built with Eclipse Temurin OpenJDK 21.0.7, compatible with Java 8 and above.

[c] Tested with Python 3.8-3.12.

[d] Built with .NET 6, compatible with .NET 5+ (and newer). Also supported on .NET Core 2.1 (and above) and .NET Framework 4.8.3 (and above).

[e] Tested with C++11 unless stated otherwise.

[f] Supporting Modern C++ means supporting the RPC feature unless otherwise stated.

[g] Advanced example generation in code generator not supported.

[h] Advanced example generation in code generator not supported.

### 3.1 Summary of Android Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Transports							
	UDP/IPv4	UDP/IPv6	Multicast	TCP/IPv4	Shared Memory (w/ zero copy) <sup>[b]</sup>		
Android 12, 14 arm64Android12clang12.0.8ndkr23b	•	•	•	•			
Android AOSP 14 arm64AndroidAOSP14clang17.0.2	•	•	•				
Infrastructure Services							
	Persistence Service <sup>[c]</sup>	Routing Service	Recording and Replay Services	Web Integration Service			
Android 12, 14 arm64Android12clang12.0.8ndkr23b							
Android AOSP 14 arm64AndroidAOSP14clang17.0.2							
Connex Professional Libraries							
	Request/Reply	Monitoring Library	Distributed Logger	LBED <sup>[d]</sup>	Monitoring Library 2.0		
Android 12, 14 arm64Android12clang12.0.8ndkr23b	•	•	•		See "Observability Framework" rows below		
Android AOSP 14 arm64AndroidAOSP14clang17.0.2	•	•	•				

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Zero-copy transfer over shared memory is NOT supported for Java, Ada, .NET and Python programming languages. It is only supported for C, Traditional C++, and Modern C++ programming languages.

<sup>[c]</sup> Tested with filesystem only in PERSISTENT mode.

<sup>[d]</sup> Supports dynamic linking only.

### 3.1 Summary of Android Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Core Library Features							
	Monotonic Clock	CPU Core Affinity	Durable writer history & reader state	RTI DDS thread name	Backtrace	Cmake Find Package	
Android 12, 14 arm64Android12clang12.0.8ndkr23b	•						
Android AOSP 14 arm64AndroidAOSP14clang17.0.2	•						
Tools							
	Shapes Demo	Launcher	Monitor	Admin Console	System Designer	rtiddsgen server	Utilities (rtiddsping, rtiddsspy)
Android 12, 14 arm64Android12clang12.0.8ndkr23b							•
Android AOSP 14 arm64AndroidAOSP14clang17.0.2							•
Observability Framework							
	Observability Collector Service	Monitoring Library 2.0					
Android 12, 14 arm64Android12clang12.0.8ndkr23b		•					
Android AOSP 14 arm64AndroidAOSP14clang17.0.2		•					

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

RTI Architecture <sup>[a]</sup>	Supported Features						
Security Extensions							
	Security Plugins (for OpenSSL) <sup>[b]</sup>	Security Plugins (for wolfSSL) <sup>[c]</sup>	TLS Support <sup>[d]</sup>	Security Plugins SDK <sup>[e]</sup> , <sup>[f]</sup>			
Android 12, 14 arm64Android12clang12.0.8ndkr23b	•		•				
Android AOSP 14 arm64AndroidAOSP14clang17.0.2							
Add-ons							
	Cloud Discovery Service	Real-Time WAN Transport	Ada Language Support	Limited Bandwidth Plugins	Queuing Service (Experimental)		
Android 12, 14 arm64Android12clang12.0.8ndkr23b		•					
Android AOSP 14 arm64AndroidAOSP14clang17.0.2		•					

### 3.1.1 Monotonic Clock Support

See [Configuring the Clock per DomainParticipant](#), in the *RTI Connex Core Libraries User's Manual* for information on the monotonic clock.

## 3.2 Building Applications for Android Platforms

### 3.2.1 Required Libraries and Compiler Flags

First, see the basic instructions in [Chapter 2 Building Applications—Notes for All Platforms](#) on page 4.

See [Table 3.2 Building Instructions for Android Architectures](#) for a list of the compiler flags and libraries you will need to link into your application.

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[c]</sup> Tested with WolfSSL 5.5.1.

<sup>[d]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[e]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[f]</sup> Tested with WolfSSL 5.5.1.

Depending on which *Connex* features you want to use, you may need additional libraries; see [3.2.2 Additional Libraries for Other Features](#) on page 16.

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

### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*, *debug* libraries are created with debug symbols to facilitate debugging with **debugger**, for example. *Release* libraries do not contain debug information.

The *Connex* debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

### Additional Documentation:

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

**Table 3.2 Building Instructions for Android Architectures**

API	Library Format	Required RTI Libraries and JAR Files <sup>[a]</sup> , [b]	Required System Libraries	Required Compiler Flags
C++ [c]	Release	libnndscore.so libnndsc.so libnndscpp.so (or libnndscpp2.so) librticonnextmsgcpp.so libc++_shared.so	<b>For Android 12, 14:</b> -L\$(ANDROID_NDK_ROOT)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/aarch64-linux-android -lc++_shared	<b>For Android 12, 14:</b> -DRTI_LINUX -DRTI_UNIX -DRTI_64BIT -DRTI_ANDROID=12  <b>For AOSP 14:</b> -Wno-return-type-c-linkage -Wno-deprecated-register -nostdlibinc -march=armv8-a -target aarch64-linux-android -I\${AOSP_ROOT}/external/libcxx/include -I\${AOSP_ROOT}/external/libcxxabi/include -isystem \${AOSP_ROOT}/bionic/libc/include -isystem \${AOSP_ROOT}/bionic/libc/kernel/uapi -isystem \${AOSP_ROOT}/bionic/libc/kernel/android/uapi -isystem \${AOSP_ROOT}/bionic/libc/kernel/uapi/asm-arm64 -I\${AOSP_ROOT}/system/logging/liblog/include -I\${AOSP_ROOT}/libnativehelper/include_jni -fpic -DLINUX -DRTI_LINUX -DRTI_ANDROID=14 -DRTI_AOSP=14 -DRTI_64BIT -DRTI_LINUX26 -DRTI_UNIX
	Debug	libnndscored.so libnndscd.so libnndscppd.so (or libnndscpp2d.so) librticonnextmsgcppd.so libc++_shared.so	<b>For AOSP 14:</b> See <a href="#">3.2.1.1 Linking executables for C++</a> on page 15 and <a href="#">3.2.1.3 Building shared libraries instead of executables (C and C++)</a> on page 15	

[a] Choose libnndscpp\*. \* for the Traditional C++ API or libnndscpp2\*. \* for the Modern C++ API.

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

[c] Traditional and Modern APIs)

Table 3.2 Building Instructions for Android Architectures

API	Library Format	Required RTI Libraries and JAR Files <sup>[a]</sup> , <sup>[b]</sup>	Required System Libraries	Required Compiler Flags
C	Release	libnddscore.so libnddsc.so librticonnextmsgc.so	<p><b>For Android 12, 14:</b></p> <p>-L\$(ANDROID_NDK_ROOT)/toolchains/llvm/prebuilt/linux-x86_64/sysroot/usr/lib/aarch64-linux-android/31 -llog -lc -lm</p> <p><b>For AOSP 14:</b></p> <p>See <a href="#">3.2.1.2 Linking executables for C on the next page</a> and <a href="#">3.2.1.3 Building shared libraries instead of executables (C and C++) on the next page</a></p>	<p><b>For Android 12, 14:</b></p> <p>-DRTI_LINUX -DRTI_UNIX -DRTI_64BIT -DRTI_ANDROID=12</p> <p><b>For AOSP 14:</b></p> <p>-nostdlibinc -march=armv8-a -target aarch64-linux-android -fpic -DLINUX -DRTI_LINUX -DRTI_ANDROID=14 -DRTI_AOSP=14 -DRTI_64BIT -DRTI_LINUX26 -DRTI_UNIX -isystem \${AOSP_ROOT}/bionic/libc/include -isystem \${AOSP_ROOT}/bionic/libc/kernel/uapi -isystem \${AOSP_ROOT}/bionic/libc/kernel/android/uapi -isystem \${AOSP_ROOT}/bionic/libc/kernel/uapi/asm-arm64 -I\${AOSP_ROOT}/system/logging/liblog/include -I\${AOSP_ROOT}/libnativehelper/include_jni</p>
	Debug	libnddscored.so libnddscd.so librticonnextmsgcd.so		
Java	Release	<p><b>When not building Apps (*.apk):</b></p> <p>nddsjava.jar rticonnextmsg.jar</p> <p><b>When building Apps (*.apk):</b></p> <p>nddsjava.jar libnddsjava.so libnddscore.so libnddsc.so rticonnextmsg.jar</p>	N/A	None required
	Debug	<p><b>When not building Apps (*.apk):</b></p> <p>nddsjavad.jar rticonnextmsgd.jar</p> <p><b>When building Apps (*.apk):</b></p> <p>nddsjavad.jar libnddsjavad.so libnddscored.so libnddscd.so rticonnextmsgd.jar</p>		

[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>.

### 3.2.1.1 Linking executables for C++

1. Prepend the following before the output executable:

```
-fuse-ld=lld -nostdlib -target aarch64-linux-android ${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/crtbegin_dynamic/android_arm64_armv8-a/crtbegin_dynamic.o -L${AOSP_ROOT}/out/soong/.intermediates/external/libcxx/libc++/android_arm64_armv8-a_shared -lc++ -L${AOSP_ROOT}/out/soong/.intermediates/system/logging/liblog/liblog/android_arm64_armv8-a_shared -llog -L${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/libc/android_arm64_armv8-a_shared -lc -L${AOSP_ROOT}/out/soong/.intermediates/bionic/libm/libm/android_arm64_armv8-a_shared -lm
```

2. Add the following after the output executable:

```
${AOSP_ROOT}/out/soong/.intermediates/external/libcxxabi/libc++demangle/android_arm64_armv8-a-static/libc++demangle.a -Wl,--start-group ${AOSP_ROOT}/prebuilts/clang/host/linux-x86/clang-r487747c/lib/clang/17.0.2/lib/linux/libclang_rt.builtins-aarch64-android.a -Wl,--end-group ${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/crtend_android/android_arm64_armv8-a/obj/bionic/libc/arch-common/bionic/crtend.o -Wl,--exclude-libs=libclang_rt.builtins-aarch64-android.a
```

### 3.2.1.2 Linking executables for C

1. Prepend the following before the output executable:

```
-fuse-ld=lld -nostdlib -target aarch64-linux-android ${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/crtbegin_dynamic/android_arm64_armv8-a/crtbegin_dynamic.o -L${AOSP_ROOT}/out/soong/.intermediates/system/logging/liblog/liblog/android_arm64_armv8-a_shared -llog -L${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/libc/android_arm64_armv8-a_shared -lc -L${AOSP_ROOT}/out/soong/.intermediates/bionic/libm/libm/android_arm64_armv8-a_shared -lm
```

2. Add the following after the output executable:

```
${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/crtend_android/android_arm64_armv8-a/obj/bionic/libc/arch-common/bionic/crtend.o
```

### 3.2.1.3 Building shared libraries instead of executables (C and C++)

If building shared libraries rather than executables, use the linking instructions for executables above for the appropriate language, and instead of linking against `crtbegin_dynamic.o` (the first object to link, before the output executable) link with: `${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/crtbegin_so/android_arm64_armv8-a/crtbegin_so.o`

Likewise, rather than linking against `crtend.o` (the last object to link, after the output executable) link with: `${AOSP_ROOT}/out/soong/.intermediates/bionic/libc/crtend_so/android_arm64_armv8-a/obj/bionic/libc/arch-common/bionic/crtend_so.o`

## 3.2.2 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to [Table 3.1 Android Systems - Supported Features](#) to see if these libraries are supported on your platform.

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 library. Do not mix release and debug libraries.

### 3.2.2.1 Libraries Required for Distributed Logger

*RTI Distributed Logger* is supported on all the platforms in [Chapter 3 Android Platforms](#).

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

**Table 3.3 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

### 3.2.2.2 Libraries Required for Monitoring

**Table 3.4 Additional Libraries for Using Monitoring**

Library Format	Monitoring Libraries <sup>[a]</sup>
Release	librtimonitoring.so
Debug	librtimonitoringd.so

### 3.2.2.3 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 3.5 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 3.5 Additional Libraries for Using RTI Real-Time WAN Transport APIs**

Library Format	Real-Time WAN Transport Libraries <sup>[a]</sup>
Release	libnndsrwt.so
Debug	libnndsrwtd.so

### 3.2.2.4 Libraries Required for TCP Transport and TLS Support

These transports are supported on Android 12 and 14, but not AOSP 14.

To use the TCP Transport APIs, link against the additional libraries in [Table 3.6 Additional Libraries for Using RTI TCP Transport APIs](#). If you are using *RTI TLS Support*, also link against the libraries in [Table 3.7 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>.

**Table 3.6 Additional Libraries for Using RTI TCP Transport APIs**

Library Format	TCP Transport Libraries <sup>[a]</sup>
Release	libnndstransporttcp.so
Debug	libnndstransporttcpd.so

**Table 3.7 Additional Libraries for Using RTI TCP Transport APIs with TLS Enabled**

Library Format	TCP Transport Libraries <sup>[b]</sup>	OpenSSL Libraries <sup>[c]</sup>
Release	libnndstls.so	librtisslsupport.so
Debug	libnndstlsd.so	

### 3.2.3 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 [Chapter 3 Android Platforms](#)). The build process is similar to other Linux variants, see [5.2 Building Applications for Linux Platforms on page 41](#).

### 3.2.4 '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

[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-3.5.1/<architecture>/<format>/lib.

means libraries created with debug symbols to facilitate debugging with gdb, for example. A "release" library does not contain debug information.

### 3.2.5 How the Connex Libraries were Built

[Table 3.8 Library-Creation Details for Android Architectures below](#) 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 [Chapter 2 Building Applications—Notes for All Platforms on page 4](#).

**Table 3.8 Library-Creation Details for Android Architectures**

RTI Architecture	Library Format	Compiler Flags Used by RTI
arm64Android12clang12.0.8ndkr23b when not using Java	Release	--target=aarch64-none-linux-android31 -DLINUX -DSIZEOF_LONG=8 -O3 -funwind-tables -no-canonical-prefixes -fexceptions -frtti -stdlib=libc++ -flto=thin -O3 -DNDEBUG -fPIC -fvisibility=hidden -fno-strict-aliasing
	Debug	--target=aarch64-none-linux-android31 -DLINUX -DSIZEOF_LONG=8 -O0 -funwind-tables -no-canonical-prefixes -fexceptions -frtti -stdlib=libc++ -g -fPIC -fno-strict-aliasing
arm64Android12clang12.0.8ndkr23b when using Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g
arm64AndroidAOSP14clang17.0.2	Dynamic Release	--target=aarch64-linux-android -DLINUX -DSIZEOF_LONG=8 -O3 -march=armv8-a -nostdlibc -flto=thin -DNDEBUG -fPIC -fvisibility=hidden -fno-strict-aliasing
	Dynamic Debug	--target=aarch64-linux-android -DLINUX -DSIZEOF_LONG=8 -O0 -march=armv8-a -nostdlibc -O0 -g -fPIC -fno-strict-aliasing

## 3.3 Running Your Applications

For the environment variables that must be set at run time, see [Table 3.9 Running Instructions for Android Architectures](#).

**Table 3.9 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-lib>

**Table 3.9 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

## 3.4 Unsupported Features

Using DDS\_WireProtocolQosPolicyAutoKind's RTPS\_AUTO\_ID\_FROM\_MAC to calculate the GUID prefix is not supported.

## 3.5 Thread Configuration

### 3.5.1 Thread Settings and Thread-Priority Definitions

See [Table 3.10 Thread Settings for Android Platforms](#) and [Table 3.11 Thread-Priority Definitions for Android Platforms](#).

**Table 3.10 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 3.10 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 3.11 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	

## 3.5.2 Automatic Thread-Specific Storage Cleanup

Android systems support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore not required to clean up thread-specific storage in user threads (although it is safe to do so).

## 3.6 Socket Buffer Size Configuration

See [Table 3.12 UDP send\\_socket\\_buffer\\_size](#) and [Table 3.13 UDP receive\\_socket\\_buffer\\_size](#). For more information on the **send\_socket\_buffer\_size** and **receive\_socket\_buffer\_size** properties, see [Setting Builtin Transport Properties with the PropertyQoSPolicy, in the RTI Connex Core Libraries User's Manual](#).

**Table 3.12 UDP send\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size <b>net.core.wmem_default</b>
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>net.core.wmem_max</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size <b>net.core.wmem_default</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>net.core.wmem_max</b>

**Table 3.13 UDP receive\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size <b>net.core.rmem_default</b>
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum receive socket buffer size <b>net.core.rmem_max</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size <b>net.core.rmem_default</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum receive socket buffer size <b>net.core.rmem_max</b>

## 3.7 Third-Party Software Versions Used for Android 12 Development and Testing

**Table 3.14 Third-Party Software for Android 12 (arm64Android12clang12.0.8ndkr23b)**

Third-Party Software	Version
Android Build Tools	35.0.0
Android Command-Line Tools	3.0
Android SDK Platforms-Tools	33
Android API Level	31
Android NDK	23.0.7243079
Gradle	8.14
Gradle Plugin	8.10

**Table 3.15 Third-Party Software for Android 14 (arm64AndroidAOSP14clang17.0.2)**

Third-Party Software	Version
Android Build Tools	35.0.0
Android Command-Line Tools	3.0
Android NDK	25.0.8474149
Gradle Plugin	8.14
Gradle	8.10

# Chapter 4 INTEGRITY Platforms

## 4.1 Building Applications for INTEGRITY Platforms

First, see the basic instructions in [Chapter 2 Building Applications—Notes for All Platforms on page 4](#).

### 4.1.1 Required Libraries and Compiler Flags

[Table 4.1 Building Instructions for INTEGRITY Architectures](#) lists the compiler flags and 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 [4.1.3 Additional Libraries for Other Features on page 26](#).

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

#### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*t, *debug* libraries are created with debug symbols to facilitate debugging with **debugger**, for example. *Release* libraries do not contain debug information.

The *Connex*t debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

Table 4.1 Building Instructions for INTEGRITY Architectures

API	Library Format	Required RTI Libraries <sup>[a]</sup> , <sup>[b]</sup> , <sup>[c]</sup>	Required System Libraries <sup>[d]</sup>	Required Compiler Flags
C++ (Traditional and Modern APIs)	Release	libniddscorez.a libniddscz.a  libniddscppz.a or libniddscpp2z.a  librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	libsocket.a libnet.a libposix.a	-DRTI_INTY --exceptions
	Debug	libniddscorezd.a libniddsczd.a  libniddscppzd.a or libniddscpp2zd.a  librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a (libniddscorezd.dba) (libniddsczd.dba) (libniddscppzd.dba or libniddscpp2zd.dba) (librticonnextmsgczd.dba)		
C	Release	libniddscorez.a libniddscz.a librticonnextmsgcz.a		
	Debug	libniddscorezd.a libniddsczd.a librticonnextmsgczd.a (libniddscorezd.dba) (libniddsczd.dba) (librticonnextmsgczd.dba)		

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

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

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

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

## 4.1.2 Required Patch for INTEGRITY 11.0.4 Platforms

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

p4080Inty11.devtree-fsl-e500mc.comp2013.5.4: **patch\_8154.iff, patch\_8155.iff, patch\_8246.iff**

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

## 4.1.3 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to the *Supported Platforms* section of the [RTI Connex Core Libraries Release Notes](#) to see if these libraries are supported on your platform.

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 library.

### 4.1.3.1 Libraries Required for Distributed Logger

[Table 4.2 Additional Libraries for using RTI Distributed Logger](#) lists the additional libraries you will need in order to use *Distributed Logger*.

**Table 4.2 Additional Libraries for using RTI Distributed Logger**

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

### 4.1.3.2 Libraries Required for Monitoring

#### Notes:

- The RTI library in [Table 4.3 Additional Libraries for Using Monitoring](#) must appear *first* in the list of libraries to be linked.

---

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

- Automatic loading of the dynamic monitoring library through QoS is not supported.
- Memory and CPU usage is not available in monitoring data.

**Table 4.3 Additional Libraries for Using Monitoring**

Library Format	Monitoring Libraries <sup>[a]</sup>
Static Release	librtimonitoringz.a
Static Debug	librtimonitoringzd.a

#### 4.1.3.3 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 using one of the libraries in [Table 4.4 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 4.4 Additional Libraries for Using RTI Real-Time WAN Transport APIs**

Library Format	Real-Time WAN Transport Libraries <sup>[b]</sup>
Static Release	libnddsrtwz.a
Static Debug	libnddsrtwzd.a

#### 4.1.3.4 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 4.5 Additional Libraries for Zero Copy Transfer Over Shared Memory](#).

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

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

**Table 4.5 Additional Libraries for Zero Copy Transfer Over Shared Memory**

Library Format	Zero Copy Transfer Over Shared Memory Libraries <sup>[a]</sup>
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

## 4.2 Running User Applications

There are no extra environment variables that must be set at run time.

## 4.3 Thread Configuration

### 4.3.1 Thread Settings and Thread-Priority Definitions

See these tables:

- [Table 4.6 Thread Settings for INTEGRITY Platforms](#)
- [Table 4.7 Thread-Priority Definitions for INTEGRITY Platforms](#)

**Table 4.6 Thread Settings for INTEGRITY Platforms**

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Asynchronous Publisher, Asynchronous flushing thread	mask	OS default thread type
	priority	16
	stack_size	64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Database thread	mask	DDS_THREAD_SETTINGS_STDIO
	priority	60
	stack_size	64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

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

Table 4.6 Thread Settings for INTEGRITY Platforms

Applicable Thread	DDS_ ThreadSettings_t	Platform-Specific Setting
Event thread	mask	DDS_THREAD_SETTINGS_STUDIO   DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	80
	stack_size	4*64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STUDIO
	priority	100
	stack_size	4*64*1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 4.7 Thread-Priority Definitions for INTEGRITY Platforms

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

### 4.3.2 Socket-Enabled and POSIX-Enabled Threads are Required

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

Furthermore, threads calling *Connex* must be socket-enabled. This can be achieved by calling **InitLibSocket()** before making any *Connex* calls and calling **ShutdownLibSocket** before the thread

terminates. Note that an Initial thread is, by default, socket-enabled when the address space is linked with **libsocket.a**. Please refer to the *INTEGRITY Development Guide* for more information.

### 4.3.3 Automatic Thread-Specific Storage Cleanup

INTEGRITY systems support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore not required to clean up thread-specific storage in user threads (although it is safe to do so).

## 4.4 Socket Buffer Size Configuration

See [Table 4.8 UDP send\\_socket\\_buffer\\_size](#) and [Table 4.9 UDP receive\\_socket\\_buffer\\_size](#). For more information on the **send\_socket\_buffer\_size** and **receive\_socket\_buffer\_size** properties, see [Setting Builtin Transport Properties with the PropertyQosPolicy, in the RTI Connex Core Libraries User's Manual](#).

**Table 4.8 UDP send\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses the default send socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT will be used)
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses the default send socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT will be used)

**Table 4.9 UDP receive\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses the default receive socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT will be used)
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses the default receive socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT will be used)

## 4.5 Running over IP Backplane on a Dy4 Champ-AVII Board

*Connex* can run on all four CPUs, provided the following hold true:

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

## 4.6 Out-of-the-box Transport Compatibility with Other Connex Platforms

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

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

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

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

### 4.6.1 Smaller Shared-Memory Receive-Resource Queue Size

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

To allow more *DomainParticipants* to run within the same address space, we reduced the default size of the queue for each receive resource of the shared memory transport. The queue size is reduced to

eight messages. This change only applies to INTEGRITY architectures and this default value can be overwritten through the shared memory transport QoS.

## 4.6.2 Using Shared Memory on INTEGRITY Systems

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

To clean up shared memory resources, reboot the kernel.

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

*Connex* will run on a target without the **posix\_shm\_manager**, but the POSIX functions will fail and return **ENOSYS**, and the participants will fail to communicate through shared memory.

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

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

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

```
-os_dir=/local/applications/integrity/integrity-10.0.2
```

2. Build the project.
3. Before running your *Connex* application on a target, download the **posix\_shm\_manager** file (generated by the build) onto the target.

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

### Notes:

- Only *one* **posix\_shm\_manager** is needed on a particular target. INTEGRITY offers the option of building this **posix\_shm\_manager** *inside* the kernel. Please refer to the INTEGRITY documentation.
- If you are already using shared memory through the POSIX library, there may be a possible conflict.

- INTEGRITY has two different types of POSIX libraries: a single-address space one (or 'light') and another one (complete POSIX implementation). *Connex* uses the first one, but will work if you are using the complete POSIX implementation.

### 4.6.3 Shared Memory Limitations on INTEGRITY Systems

If several applications are running on the same INTEGRITY node and are using shared memory, once an application is stopped, it cannot be restarted. When the application is stopped (gracefully or ungracefully), any new application on the same domain index within the same DDS domain will fail to start until the shared memory manager is also restarted.

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

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

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

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

## 4.7 Using *rtiddsping* and *rtiddsspy* on INTEGRITY Systems

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

## 4.8 Issues with INTEGRITY Systems

### 4.8.1 Delay When Writing to Unreachable Peers

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

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

## 4.8.2 Linking with 'libivfs.a' without a File System

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

## 4.8.3 Compiler Warnings Regarding Unrecognized #pragma Directives

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

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

"C:/ndds/ndds.4.4x/include/ndds/dds_c/dds_c_infrastructure.h", line 927:
warning: unrecognized #pragma
    #pragma warning(disable:4190)
        ^

"C:/ndds/ndds.4.4x/include/ndds/dds_c/dds_c_infrastructure.h", line 945:
warning: unrecognized #pragma
    #pragma warning(pop)
        ^
```

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

# Chapter 5 Linux Platforms

## 5.1 Summary of Linux Platforms and Supported Features

The following tables show the supported products/features for each architecture. A ● means supported in this release, an empty cell means unsupported in this release.

**Table 5.1 Linux Systems - Supported Features**

RTI Architecture <sup>[a]</sup>	Supported Features						
Languages							
	Java <sup>[b]</sup>	Python <sup>[c]</sup>	.NET (C#) <sup>[d]</sup>	C/C++	Modern C++ <sup>[e]</sup> , <sup>[f]</sup>		
Linux® (Intel) Red Hat® Enterprise Linux 8, 9; Ubuntu® 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	●	●	●	●	●		
Linux (Arm®) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0	●	●	●	●	●		

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Built with Eclipse Temurin OpenJDK 21.0.7, compatible with Java 8 and above. Also tested with AdoptOpenJDK 11.0.13 and 17.0.6.

<sup>[c]</sup> Tested with Python 3.8-3.12.

<sup>[d]</sup> Built with .NET 6, compatible with .NET 5+ (and newer). Also supported on .NET Core 2.1 (and above) and .NET Framework 4.8.3 (and above).

<sup>[e]</sup> Tested with C++11 unless stated otherwise.

<sup>[f]</sup> Supporting Modern C++ means supporting the RPC feature unless otherwise stated.

## 5.1 Summary of Linux Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_ GP					•		
Transports							
	UDP/IPv4	UDP/IPv6	Multicast	TCP/IPv4	Shared Memory (w/ zero copy) <sup>[b]</sup>		
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•	•		
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0	•	•	•	•	•		
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_ GP	•	•	•		•		
Infrastructure Services							
	Persistence Service <sup>[c]</sup>	Routing Service	Recording and Replay Services	Web Integration Service			
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•			
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0	•	•	•	•			
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_ GP							

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Zero-copy transfer over shared memory is NOT supported for Java, Ada, .NET and Python programming languages. It is only supported for C, Traditional C++, and Modern C++ programming languages.

<sup>[c]</sup> Tested with filesystem only in PERSISTENT mode.

## 5.1 Summary of Linux Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Connex Professional Libraries							
	Request/Reply	Monitoring Library	Distributed Logger	LBED <sup>[b]</sup>	Monitoring Library 2.0		
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•	See "Observability Framework" rows below		
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0	•	•	•	•			
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_GP							
Core Library Features							
	Monotonic Clock	CPU Core Affinity	Durable writer history & reader state	RTI DDS thread name	Backtrace <sup>[c]</sup>	Cmake Find Package	
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•	• <sup>[d]</sup>	•	
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0	•	•	•	•	• <sup>[e]</sup>		
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_GP	•	•		•			

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Supports dynamic linking only.

<sup>[c]</sup> If you want backtrace information: on all Linux architectures, compile with `-fno-omit-frame-pointer` and add linker flag `-rdynamic`. For Linux architectures on Arm CPUs, also compile with `"-funwind-tables"`. On all Windows architectures, you need the `Dbghelp.dll` and `NtDll.dll` libraries.

<sup>[d]</sup> For architectures using `gcc 6` or higher: to display backtrace information, you must link the executable with `"-no-pie"` to prevent PIE generation.

<sup>[e]</sup> For architectures using `gcc 6` or higher: to display backtrace information, you must link the executable with `"-no-pie"` to prevent PIE generation.

## 5.1 Summary of Linux Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Tools							
	Shapes Demo	Launcher	Monitor	Admin Console	System Designer	rtiddsgen server	Utilities (rtiddsping, rtiddsspy)
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•	• [b]	•	•
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0							•
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_GP							
Observability Framework							
	Observability Collector Service	Monitoring Library 2.0					
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•					
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0		•					
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_GP							

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Tested on x64 Linux, with Chrome version 112 and Firefox version 108.

## 5.1 Summary of Linux Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Security Extensions							
	Security Plugins (for OpenSSL) [b]	Security Plugins (for wolfSSL) [c]	TLS Support [d]	Security Plugins SDK [e] , [f]			
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•			
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0	•		•				
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_ GP							
Add-ons							
	Cloud Discovery Service	Real-Time WAN Transport	Ada Language Support	Limited Bandwidth Plugins	Queuing Service (Experimental)		
Linux (Intel) Red Hat Enterprise Linux 8, 9; Ubuntu 22.04 LTS, 24.04 LTS x64Linux4gcc8.5.0	•	•	•	•	•		
Linux (Arm) Ubuntu 22.04 LTS, 24.04 LTS armv8Linux4gcc8.5.0		•		•			
Linux (Intel) Red Hat Enterprise Linux 8 x64Linux4gcc8.5.0FACE_ GP							

See [Supported Platforms](#) and [Customized Installation](#) in *RTI Connex for Debian Linux* for more information on Debian support. See <https://hub.docker.com/u/rticom> (click the product, such as *DDS Spy*, then

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Tested with OpenSSL3.5.1 unless stated otherwise.

[c] Tested with WolfSSL 5.5.1.

[d] Tested with OpenSSL3.5.1 unless stated otherwise.

[e] Tested with OpenSSL3.5.1 unless stated otherwise.

[f] Tested with WolfSSL 5.5.1.

the Tags tab) for the Linux architectures that Docker supports; see also [rticonnextdds-containers](#) on GitHub.

## 5.1.1 Monotonic Clock Support

See [Configuring the Clock per DomainParticipant](#), in the RTI Connex Core Libraries User's Manual for information on the monotonic clock.

## 5.1.2 Support for 'Find Package' CMake Script

For information on using the 'Find Package' CMake script, see [2.5 Building with CMake on page 8](#).

## 5.1.3 Backtrace Support

Note the following for platforms that support the backtrace feature:

- 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 Connex Core Libraries User's Manual.

## 5.1.4 Limitations of FACE Architectures

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

- x64Linux4gcc8.5.0FACE\_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`.

In [Table 5.1 Linux Systems - Supported Features](#), blank cells for x64Linux4gcc8.5.0FACE\_GP indicate that the feature, utility, or tool is not supported by the FACE architecture.

## 5.2 Building Applications for Linux Platforms

First, see the basic instructions in [Chapter 2 Building Applications—Notes for All Platforms on page 4](#).

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/](http://www.gnu.org/software/make/) or download from [ftpmirror.gnu.org/make](http://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>
```

### 5.2.1 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. [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 [5.2.2 Additional Libraries for Other Features on page 43](#)

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.

#### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*, *debug* libraries are created with debug symbols to facilitate debugging with **gdb**, for example. *Release* libraries do not contain debug information.

The *Connex* debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional

diagnostic information that can affect runtime behavior. They are not intended for use in production environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

**Table 5.2 Building Instructions for Linux Architectures**

API	Library Format	Required RTI Libraries or Jar Files <sup>[a]</sup> , <sup>[b]</sup>	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libniddscorez.a libniddscz.a  libniddscppz.a or libniddscpp2z.a  librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-ldl -lm -lpthread -lrt	<p>For 64-bit architectures: -DRTI_LINUX -DRTI_UNIX -m64 -DRTI_64BIT</p> <p>For any Linux platform with GCC 6 or higher linker flag (see <b>Note</b> below table), also add: -no-pie</p> <p>For all architectures, if you want backtrace information, also add: Compiler flag: -fno-omit-frame-pointer Linker flag: -rdynamic Arm architectures: -funwind-tables (see <a href="#">5.1.3 Backtrace Support on page 40</a>)</p>
	Static Debug	libniddscorezd.a libniddsczd.a  libniddscppzd.a or libniddscpp2zd.a  librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libniddscore.so libniddsc.so  libniddscpp.so or libniddscpp2.so  librticonnextmsgcpp.so or librticonnextmsgcpp2.so		
	Dynamic Debug	libniddscored.so libniddscd.so  libniddscppd.so or libniddscpp2d.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 <sup>[a]</sup> , <sup>[b]</sup>	Required System Libraries	Required Compiler Flags
C	Static Release	libniddscorez.a libniddscz.a librticonnextmsgcz.a	-ldl -lm -lpthread -lrt	For 64-bit architectures:  -DRTI_LINUX -DRTI_UNIX -m64 -DRTI_64BIT
	Static Debug	libniddscorezd.a libniddsczd.a librticonnextmsgczd.a		For any Linux platform with GCC 6 or higher linker flag (see <b>Note</b> below table), also add:  -no-pie
	Dynamic Release	libniddscore.so libniddsc.so librticonnextmsgc.so		For all architectures, if you want backtrace information, also add:  Compiler flag: -fno-omit-frame-pointer Linker flag: -rdynamic (see <a href="#">5.1.3 Backtrace Support on page 40</a> )
	Dynamic Debug	libniddscored.so libniddscd.so librticonnextmsgcd.so		
Java	Release	niddsjava.jar rticonnextmsg.jar	N/A	None required
	Debug	niddsjavad.jar rticonnextmsgd.jar		

**Note:**

For Linux platforms with GCC 6 or higher, it's possible to configure the compiler driver to link, by default, executables with PIE (position independent executable) support on amd64 and ppc64el architectures. Depending on the distributor of the GCC package, automatic PIE generation may or may not be enabled.

To correctly generate backtraces, PIE executables cannot be used with RTI's libraries. This is due to Address Space Layout Randomization (ASLR), which prevents the correct generation of backtraces of our binaries on certain systems. For this reason, RTI has linked Linux executables using the **-no-pie** flag when the GCC version is 6 or higher.

If you are using GCC 6 or higher, you must link the executable with **-no-pie** to prevent PIE generation and to correctly generate backtraces.

## 5.2.2 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to [Table 5.1 Linux Systems - Supported Features](#) to see if these libraries are supported on your platform.

[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`.

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 library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

### 5.2.2.1 Libraries Required for Distributed Logger

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

### 5.2.2.2 Libraries Required for Monitoring

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 below](#) must appear *first* in the list of libraries to be linked.

**Table 5.4 Additional Libraries for Using Monitoring**

Library Format	Monitoring Libraries <sup>[a]</sup>
Dynamic Release	librtimonitoring.so
Dynamic Debug	librtimonitoringd.so

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

**Table 5.4 Additional Libraries for Using Monitoring**

Library Format	Monitoring Libraries <sup>[a]</sup>
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

### 5.2.2.3 Libraries Required for Real-Time WAN Transport

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

Using *Real-Time WAN Transport* requires one of the libraries in [Table 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 <sup>[b]</sup>
Dynamic Release	libnddsrtw.so
Dynamic Debug	libnddsrtwd.so
Static Release	libnddsrtwz.a
Static Debug	libnddsrtwdz.a

### 5.2.2.4 Libraries Required for TCP Transport and TLS Support

To use the TCP Transport APIs, link against the additional libraries in [Table 5.6 Additional Libraries for using RTI TCP Transport APIs below](#).

**Table 5.6 Additional Libraries for using RTI TCP Transport APIs**

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

**Table 5.6 Additional Libraries for using RTI TCP Transport APIs**

Library Format	RTI TCP Transport Libraries <sup>[a]</sup>
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 *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 <sup>[b]</sup>	OpenSSL Libraries <sup>[c]</sup>
Dynamic Release	libniddstls.so	libssl.so libcrypto.so
Dynamic Debug	libniddstlsd.so	
Static Release	libniddstlsz.a	
Static Debug	libniddstlszd.a	

### 5.2.2.5 Libraries Required for Zero Copy Transfer Over Shared Memory

The Zero Copy Transfer Over Shared Memory feature is supported on all the platforms in [Chapter 5 Linux Platforms on page 35](#).

To use this feature, link against the additional library in [Table 5.8 Additional Libraries for Zero Copy Transfer Over Shared Memory below](#).

**Table 5.8 Additional Libraries for Zero Copy Transfer Over Shared Memory**

Library Format	Zero Copy Transfer Over Shared Memory Libraries <sup>[d]</sup>
Dynamic Release	libniddsmetp.so
Dynamic Debug	libniddsmetpd.so

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

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

[c] OpenSSL libraries are in <NDDSHOME>/third\_party/openssl-3.5.1/<architecture>/<format>/lib.

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

**Table 5.8 Additional Libraries for Zero Copy Transfer Over Shared Memory**

Library Format	Zero Copy Transfer Over Shared Memory Libraries <sup>[a]</sup>
Static Release	libniddsmetpz.a
Static Debug	libniddsmetpzd.a

## 5.2.3 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

### 5.2.3.1 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
x64Linux4gcc8.5.0	x64	2.28	6.0.25 <= X
armv8Linux4gcc8.5.0	Arm v8	2.28	6.0.25 <= X
x64Linux4gcc8.5.0FACE_GP	x64	2.28	6.0.25 <= X

### 5.2.3.2 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.

### 5.2.3.3 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.

## 5.2.4 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
armv8Linux4gcc8.5.0	Static Release	-O3 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -fno-strict-aliasing -fno-semantic-interposition -O3 -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="armv8Linux4gcc8.5.0" -Werror=implicit-function-declaration
	Static Debug	-O0 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="armv8Linux4gcc8.5.0" -Werror=implicit-function-declaration
	Dynamic Release	-O3 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -fno-strict-aliasing -fno-semantic-interposition -O3 -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="armv8Linux4gcc8.5.0" -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -funwind-tables -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="armv8Linux4gcc8.5.0" -Werror=implicit-function-declaration
x64Linux4gcc8.5.0	Static Release	-O3 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O3 -DNDEBUG -fno-strict-aliasing -fno-semantic-interposition -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc8.5.0" -Werror=implicit-function-declaration
	Static Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc8.5.0" -Werror=implicit-function-declaration
	Dynamic Release	-O3 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -fno-strict-aliasing -fno-semantic-interposition -O3 -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc8.5.0" -Werror=implicit-function-declaration
	Dynamic Debug	-O0 -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="x64Linux4gcc8.5.0" -Werror=implicit-function-declaration

Table 5.10 Library-Creation Details for Linux Architectures

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Linux4gcc8.5.0FACE_GP	Static Release	-O -m64 -Wall -Wno-unknown-pragmas -fno-omit-frame-pointer -O -DNDEBUG -fPIC -fno-strict-aliasing -fno-semantic-interposition -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc8.5.0FACE_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 -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc8.5.0FACE_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 -fno-strict-aliasing -fno-semantic-interposition -O -DNDEBUG -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc8.5.0FACE_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 -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC -DLINUX -DPtrIntType=long -DTARGET="\x64Linux4gcc8.5.0FACE_GP" -DFACE_COMPLIANCE_LEVEL_GENERAL=4 -DENABLE_FACE_COMPLIANCE=FACE_COMPLIANCE_LEVEL_GENERAL -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

## 5.3 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} <b>Note:</b> For all 64-bit Java architectures (...64Linux...), use <b>-d64</b> 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}

## 5.3.1 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.

## 5.4 Thread Configuration

### 5.4.1 Thread Settings, Thread-Priority Definitions, and Thread Kinds

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

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

**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

**Table 5.12 Thread Settings for Linux Platforms**

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
Event thread	mask	DDS_THREAD_SETTINGS_STDIO   DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	Empty CPU list
	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 <sup>[a]</sup>
DDS_THREAD_SETTINGS_FLOATING_POINT	N/A
DDS_THREAD_SETTINGS_STDIO	N/A

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

**Table 5.14 Thread Kinds for Linux Platforms**

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

## 5.4.2 Using REALTIME\_PRIORITY

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

## 5.4.3 Automatic Thread-Specific Storage Cleanup

Linux systems support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore not required to clean up thread-specific storage in user threads (although it is safe to do so).

## 5.5 Socket Buffer Size Configuration

See [Table 5.15 UDP send\\_socket\\_buffer\\_size](#) and [Table 5.16 UDP receive\\_socket\\_buffer\\_size](#). For more information on the **send\_socket\_buffer\_size** and **receive\_socket\_buffer\_size** properties, see [Setting Builtin Transport Properties with the PropertyQosPolicy, in the RTI Connext Core Libraries User's Manual](#).

<sup>[a]</sup> See the Linux programmer's manuals for more information.

**Table 5.15 UDP send\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size <b>net.core.wmem_default</b>
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>net.core.wmem_max</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size <b>net.core.wmem_default</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>net.core.wmem_max</b>

**Table 5.16 UDP receive\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size <b>net.core.rmem_default</b>
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum receive socket buffer size <b>net.core.rmem_max</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size <b>net.core.rmem_default</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum receive socket buffer size <b>net.core.rmem_max</b>

# Chapter 6 macOS Platforms

## 6.1 Summary of macOS Platforms and Supported Features

The following table shows the supported products/features for each architecture. A ● means supported in this release, an empty cell means unsupported in this release.

**Table 6.1 macOS Systems - Supported Features**

RTI Architecture <sup>[a]</sup>	Supported Features						
Languages							
	Java <sup>[b]</sup>	Python <sup>[c]</sup>	.NET (C#) <sup>[d]</sup>	C/C++	Modern C++ <sup>[e]</sup> , <sup>[f]</sup>		
macOS® 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	●	●	●	●	●		

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Built with Eclipse Temurin OpenJDK 21.0.7, compatible with Java 8 and above. Also tested with AdoptOpenJDK 11.0.13 and 17.0.6.

[c] Tested with Python 3.8-3.12.

[d] Built with .NET 6, compatible with .NET 5+ (and newer). Also supported on .NET Core 2.1 (and above) and .NET Framework 4.8.3 (and above).

[e] Tested with C++11 unless stated otherwise.

[f] Supporting Modern C++ means supporting the RPC feature unless otherwise stated.

## 6.1 Summary of macOS Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Transports							
	UDP/IPv4	UDP/IPv6	Multicast	TCP/IPv4	Shared Memory (w/ zero copy) <sup>[b]</sup>		
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	•	•	•	•	•		
Infrastructure Services							
	Persistence Service <sup>[c]</sup>	Routing Service	Recording and Replay Services	Web Integration Service			
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	•	•	•	•			
Connex Professional Libraries							
	Request/Reply	Monitoring Library	Distributed Logger	LBED <sup>[d]</sup>	Monitoring Library 2.0		
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	•	•	•		See "Observability Framework" rows below		
Core Library Features							
	Monotonic Clock	CPU Core Affinity	Durable writer history & reader state	RTI DDS thread name	Backtrace	Cmake Find Package	
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0			•	•	•	•	

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Zero-copy transfer over shared memory is NOT supported for Java, Ada, .NET and Python programming languages. It is only supported for C, Traditional C++, and Modern C++ programming languages.

[c] Tested with filesystem only in PERSISTENT mode.

[d] Supports dynamic linking only.

## 6.1 Summary of macOS Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Tools							
	Shapes Demo	Launcher	Monitor	Admin Console	System Designer <sup>[b]</sup>	rtiddsgen server	Utilities (rtiddsping, rtiddsspy)
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	•	•	•	•	•	•	•
Observability Framework							
	Observability Collector Service	Monitoring Library 2.0					
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0		•					
Security Extensions							
	Security Plugins (for OpenSSL) <sup>[c]</sup>	Security Plugins (for wolfSSL) <sup>[d]</sup>	TLS Support <sup>[e]</sup>	Security Plugins SDK <sup>[f]</sup>			
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	•		•	•			
Add-ons							
	Cloud Discovery Service	Real-Time WAN Transport	Ada Language Support	Limited Bandwidth Plugins	Queuing Service (Experimental)		
macOS 14, 15 (Darwin 23, 24) arm64Darwin23clang16.0	•	•			•		

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Tested on ARM64 macOS, with Safari version 17.0.

<sup>[c]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[d]</sup> Tested with WolfSSL 5.5.1.

<sup>[e]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[f]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

## 6.1.1 Support for 'Find Package' CMake Script

For information on using the 'Find Package' CMake script, see [2.5 Building with CMake on page 8](#).

## 6.1.2 Backtrace Support

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

## 6.2 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 [6.2.1 Additional Libraries for Other Features on page 60](#).

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.

### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*, *debug* libraries are created with debug symbols to facilitate debugging with **lldb**, for example. *Release* libraries do not contain debug information.

The *Connex* debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

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	libnndscorez.a libnndscz.a  libnndscppz.a or libnndscpp2z.a  librticonnextmsgcppz.a or librticonnextmsgcpp2z.a	-ldl -lm -lpthread	-DRTI_UNIX -DRTI_DARWIN
	Static Debug	libnndscorezd.a libnndsczd.a  libnndscppzd.a or libnndscpp2zd.a  librticonnextmsgcppzd.a or librticonnextmsgcpp2zd.a		
	Dynamic Release	libnndscore.dylib libnndsc.dylib  libnndscpp.dylib or libnndscpp2.dylib  librticonnextmsgcpp.dylib or librticonnextmsgcpp2.dylib		
	Dynamic Debug	libnndscored.dylib libnndscd.dylib  libnndscppd.dylib or libnndscpp2d.dylib  librticonnextmsgcppd.dylib or librticonnextmsgcpp2d.dylib		

[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>/.

<NDDSHOME> is where *Connex* is installed, see [1.1 Paths Mentioned in Documentation on page 2](#)

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	-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		

## 6.2.1 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to [Table 6.1 macOS Systems - Supported Features](#) to see if these libraries are supported on your platform.

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 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>/`.

`<NDDSHOME>` is where *Connex*t is installed, see [1.1 Paths Mentioned in Documentation on page 2](#)

### 6.2.1.1 Libraries Required for Distributed Logger

[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

### 6.2.1.2 Libraries Required for Monitoring

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 <sup>[a]</sup>
Dynamic Release	librtmonitoring.dylib
Dynamic Debug	librtmonitoringd.dylib
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

### 6.2.1.3 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 [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 <sup>[b]</sup>
Dynamic Release	libnndsrwt.dylib
Dynamic Debug	libnndsrwtd.dylib
Static Release	libnndsrwtz.a
Static Debug	libnndsrwtzd.a

### 6.2.1.4 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 <sup>[a]</sup>
Dynamic Release	libnddstransporttcp.dylib
Dynamic Debug	libnddstransporttcpd.dylib
Static Release	libnddstransporttcpz.a
Static Debug	libnddstransporttcpzd.a

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

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

### 6.2.1.5 Libraries Required for Zero Copy Transfer Over Shared Memory

To use the Zero Copy Transfer Over Shared Memory feature, link against the additional library in [Table 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	libnddsmetp.dylib
Dynamic Debug	libnddsmetpd.dylib
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

## 6.2.2 How the Connex 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

[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-3.5.1/<architecture>/<format>/lib.

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
arm64Darwin23clang16.0	Release	-O3 -Wall -Wno-unknown-pragmas -Wno-trigraphs -Wmissing-field-initializers -Wuninitialized -fno-strict-aliasing -DNDEBUG -arch arm64 -mmacosx-version-min=11 -fPIC -fvisibility=hidden -Werror=implicit-function-declaration
	Debug	-O0 -Wall -Wno-unknown-pragmas -Wno-trigraphs -Wmissing-field-initializers -Wuninitialized -fno-strict-aliasing -g -arch arm64 -mmacosx-version-min=11 -fPIC -Werror=implicit-function-declaration
arm64Darwin23clang16.0 for Java	Release	-target 1.8 -source 1.8
	Debug	-target 1.8 -source 1.8 -g

## 6.3 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 <sup>[a]</sup>
arm64Darwin23clang16.0	Static	None required
	Dynamic	DYLD_LIBRARY_PATH=\${NDDSHOME}/lib/arm64Darwin23clang16.0:\${DYLD_LIBRARY_PATH}
arm64Darwin23clang16.0 for Java	N/A	DYLD_LIBRARY_PATH=\${NDDSHOME}/lib/arm64Darwin23clang16.0:\${DYLD_LIBRARY_PATH}

## 6.4 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>.

### 6.4.1 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 libnndsjava.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:

[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 `nndsjava.jar`, the Java virtual machine (JVM) will attempt to load release versions of the native libraries (`nndsjava.dylib`, `nndscore.dylib`, `nndsc.dylib`). When using `nndsjava.jar`, the JVM will attempt to load debug versions of the native libraries (`nndsjava.dylib`, `nndscore.dylib`, `nndsc.dylib`).

```
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-libndstransporttcp.dylib</value>
      </element>
      <!-- ... -->
    </value>
  </property>
</domain_participant_qos>
```

For more on transport plugins, see [6.2.1.4 Libraries Required for TCP Transport on page 62](#).

## 6.4.2 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>
```

## 6.5 Thread Configuration

### 6.5.1 Thread Settings and Thread-Priority Definitions

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_STDIO
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported
Event thread	mask	DDS_THREAD_SETTINGS_STDIO   DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

Table 6.11 Thread Settings for macOS Platforms

Applicable Thread	DDS_ThreadSettings_t	Platform-Specific Setting
ReceiverPool threads	mask	DDS_THREAD_SETTINGS_STDIO   DDS_THREAD_SETTINGS_FLOATING_POINT
	priority	OS default thread priority
	stack_size	OS default thread stack size
	cpu_list	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	

## 6.5.2 Automatic Thread-Specific Storage Cleanup

macOS systems support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore not required to clean up thread-specific storage in user threads (although it is safe to do so).

## 6.6 Socket Buffer Size Configuration

See [Table 6.13 UDP send\\_socket\\_buffer\\_size and receive\\_socket\\_buffer\\_size](#). For more information on the `send_socket_buffer_size` and `receive_socket_buffer_size` properties, see [Setting Builtin Transport Properties with the PropertyQosPolicy, in the RTI Connext Core Libraries User's Manual](#).

Table 6.13 UDP send\_socket\_buffer\_size and receive\_socket\_buffer\_size

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send and receive socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send and receive socket buffer size <b>kern.ipc.maxsockbuf</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send and receive socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send and receive socket buffer size <b>kern.ipc.maxsockbuf</b>

## 6.7 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
```

# Chapter 7 QNX Platforms

## 7.1 Summary of QNX Platforms and Supported Features

The following table shows the supported products/features for each architecture. A ● means supported in this release, an empty cell means unsupported in this release.

**Table 7.1 QNX Systems - Supported Features**

RTI Architecture [a]	Supported Features						
Languages							
	Java [b]	Python [c]	.NET (C#) [d]	C/C++	Modern C++ [e], [f]		
QNX Neutrino 8.0 [g] x64QNX8.0qcc_ cxx12.2.0 [h]				●	●		

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Built with Eclipse Temurin OpenJDK 21.0.7, compatible with Java 8 and above. Also tested with AdoptOpenJDK 11.0.13 and 17.0.6.

[c] Tested with Python 3.8-3.12.

[d] Built with .NET 6, compatible with .NET 5+ (and newer). Also supported on .NET Core 2.1 (and above) and .NET Framework 4.8.3 (and above).

[e] Tested with C++11 unless stated otherwise.

[f] Supporting Modern C++ means supporting the RPC feature unless otherwise stated.

[g] The earliest versions of the com.qnx.qnx800.target.microkernel.core (“QNX® SDP 8.0 Kernel and libc”) component are not supported. Please update this component in your QNX 8.0 SDP installation to version 2.0.2.00388T202406131303L or higher (“Stable (GA8.0.1) July 09, 2024”) before generating the QNX system image.

[h] LLVM C++ library

## 7.1 Summary of QNX Platforms and Supported Features

RTI Architecture [a]	Supported Features						
QNX Neutrino 8.0 [b] armv8QNX8.0qcc_ cxx12.2.0 [c]				•	•		
Transports							
	UDP/IPv4	UDP/IPv6	Multicast	TCP/IPv4	Shared Memory (w/ zero copy) [d]		
QNX Neutrino 8.0 x64QNX8.0qcc_ cxx12.2.0	•	• [e]	•	•	•		
QNX Neutrino 8.0 armv8QNX8.0qcc_ cxx12.2.0	•	• [f]	•	•	•		
Infrastructure Services							
	Persistence Service [g]	Routing Service	Recording and Replay Services	Web Integration Service			
QNX Neutrino 8.0 x64QNX8.0qcc_ cxx12.2.0		•					
QNX Neutrino 8.0 armv8QNX8.0qcc_ cxx12.2.0		•					

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] The earliest versions of the com.qnx.qnx800.target.microkernel.core (“QNX® SDP 8.0 Kernel and libc”) component are not supported. Please update this component in your QNX 8.0 SDP installation to version 2.0.2.00388T202406131303L or higher (“Stable (GA8.0.1) July 09, 2024”) before generating the QNX system image.

[c] LLVM C++ library

[d] Zero-copy transfer over shared memory is NOT supported for Java, Ada, .NET and Python programming languages. It is only supported for C, Traditional C++, and Modern C++ programming languages.

[e] No Traffic Class support.

[f] No Traffic Class support.

[g] Tested with filesystem only in PERSISTENT mode.

## 7.1 Summary of QNX Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Connex Professional Libraries							
	Request/Reply	Monitoring Library	Distributed Logger	LBED [b]	Monitoring Library 2.0		
QNX Neutrino 8.0 x64QNX8.0qcc_ cxx12.2.0	•	•	•	•	See "Observability Framework" rows below		
QNX Neutrino 8.0 armv8QNX8.0qcc_ cxx12.2.0	•	•	•	•			
Core Library Features							
	Monotonic Clock	CPU Core Affinity	Durable writer history & reader state	RTI DDS thread name	Backtrace [c]	Cmake Find Package	
QNX Neutrino 8.0 x64QNX8.0qcc_ cxx12.2.0	•	•		•		•	
QNX Neutrino 8.0 armv8QNX8.0qcc_ cxx12.2.0	•	•		•		•	
Tools							
	Shapes Demo	Launcher	Monitor	Admin Console	System Designer	rtiddsgen server	Utilities (rtiddsping, rtiddsspy)
QNX Neutrino 8.0 x64QNX8.0qcc_ cxx12.2.0							•
QNX Neutrino 8.0 armv8QNX8.0qcc_ cxx12.2.0							•

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Supports dynamic linking only.

[c] If you want backtrace information: on all Linux architectures, compile with `-fno-omit-frame-pointer` and add linker flag `-rdynamic`. For Linux architectures on Arm CPUs, also compile with `“-funwind-tables”`. On all Windows architectures, you need the `Dbghelp.dll` and `NtDll.dll` libraries.

## 7.1 Summary of QNX Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
<b>Observability Framework</b>							
	Observability Collector Service	Monitoring Library 2.0					
QNX Neutrino 8.0 x64QNX8.0qcc_cxx12.2.0		•					
QNX Neutrino 8.0 armv8QNX8.0qcc_cxx12.2.0		•					
<b>Security Extensions</b>							
	Security Plugins (for OpenSSL) <sup>[b]</sup>	Security Plugins (for wolfSSL) <sup>[c]</sup>	TLS Support <sup>[d]</sup>	Security Plugins SDK <sup>[e]</sup> , <sup>[f]</sup>			
QNX Neutrino 8.0 x64QNX8.0qcc_cxx12.2.0	•		•	•			
QNX Neutrino 8.0 armv8QNX8.0qcc_cxx12.2.0	•	•	•	•			
<b>Add-ons</b>							
	Cloud Discovery Service	Real-Time WAN Transport	Ada Language Support	Limited Bandwidth Plugins	Queuing Service (Experimental)		
QNX Neutrino 8.0 x64QNX8.0qcc_cxx12.2.0		•		•			
QNX Neutrino 8.0 armv8QNX8.0qcc_cxx12.2.0		•		•			

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[c]</sup> Tested with WolfSSL 5.5.1.

<sup>[d]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[e]</sup> Tested with OpenSSL3.5.1 unless stated otherwise.

<sup>[f]</sup> Tested with WolfSSL 5.5.1.

## 7.1.1 Monotonic Clock Support

See [Configuring the Clock per DomainParticipant, in the RTI Connex Core Libraries User's Manual](#) for information on the monotonic clock.

## 7.1.2 Support for 'Find Package' CMake Script

For information on using the 'Find Package' CMake script, see [2.5 Building with CMake on page 8](#).

## 7.1.3 Notes about Transports

- **Shared Memory**

The following expression can detect the two different kinds of SHMEM mutexes that RTI supports on QNX systems: posix semaphores for older QNX versions, and shared pthread mutexes for newer QNX versions, as well other shared memory resources used by *Connex* such as plain shared memory segments:

```
ls /dev/shmem/RTIOsapiSharedMemory*
```

To clean up the shared memory resources (both segments and mutexes), remove the relevant files listed in `/dev/shmem/` and `/dev/sem/`. The shared resource names used by *Connex* begin with 'RTIOsapiSharedMemory'.

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.

- **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.

## 7.2 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* features you want to use, you may need additional libraries; see [7.2.2 Additional Libraries for Other Features on page 77](#).

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.

**Release and Debug Terminology:**

Both *release* and *debug* versions of the libraries are provided. For *Connex*t, *debug* libraries are created with debug symbols to facilitate debugging with **gdb**, for example. *Release* libraries do not contain debug information.

The *Connex*t debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

**Additional Documentation:**

You should also review the QNX chapter of the [RTI Connex](#)t Core Libraries Getting Started Guide [Addendum for Embedded Systems](#).

Table 7.2 Building Instructions for QNX Architectures

API	Library Format	Required RTI Libraries <sup>[a]</sup> , <sup>[b]</sup>	Required System Libraries	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnndscorez.a libnndscz.a libnndscppz.a (or libnndscpp2z.a) librticonnextmsgcppz.a (or librticonnextmsgcpp2z.a)	-lm -lsocket	-DRTI_QNX
	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)		
C	Static Release	libnndscorez.a libnndscz.a librticonnextmsgcz.a	-lm -lsocket	For 64-bit architectures: -DRTI_64BIT For all architectures: -DRTI_QNX
	Static Debug	libnndscorezd.a libnndsczd.a librticonnextmsgczd.a		
	Dynamic Release	libnndscore.so libnndsc.so librticonnextmsgc.so		
	Dynamic Debug	libnndscored.so libnndscd.so librticonnextmsgcd.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>.

## 7.2.1 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.

## 7.2.2 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to [Table 7.1 QNX Systems - Supported Features](#) to see if these libraries are supported on your platform.

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 library. Do not link both static and dynamic libraries. Similarly, do not mix release and debug libraries.

### 7.2.2.1 Libraries Required for Distributed Logger

[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	librtidlc.a	librtidlcd.a	librtidlc.so	librtidcd.so
C++ (Traditional API)	librtidlc.a librtidlcppz.a	librtidlcd.a librtidlcppzd.a	librtidlc.so librtidlcpp.so	librtidlcd.so librtidlcppd.so

### 7.2.2.2 Libraries Required for Monitoring

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 <sup>[a]</sup>
Dynamic Release	librtmonitoring.so
Dynamic Debug	librtmonitoringd.so
Static Release	librtmonitoringz.a
Static Debug	librtmonitoringzd.a

**7.2.2.3 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 <sup>[a]</sup>
Dynamic Release	libnddsrtwt.so
Dynamic Debug	libnddsrtwtd.so
Static Release	libnddsrtwtz.a
Static Debug	libnddsrtwtd.a

#### 7.2.2.4 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 [Chapter 7 QNX Platforms on page 70](#).

**Table 7.6 Additional Libraries for using RTI TCP Transport APIs**

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

If you are using *RTI TLS Support*, also see [Table 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 <sup>[a]</sup>	OpenSSL Libraries <sup>[b]</sup>
Dynamic Release	libnndstls.so	libssl.so libcrypto.so
Dynamic Debug	libnndstlsd.so	
Static Release	libnndstlsz.a	
Static Debug	libnndstlszd.a	

### 7.2.2.5 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 <sup>[c]</sup>
Dynamic Release	libnndsmetp.so
Dynamic Debug	libnndsmetpd.so
Static Release	libnndsmetpz.a
Static Debug	libnndsmetpzd.a

## 7.2.3 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 [Chapter 2 Building Applications—Notes for All Platforms on page 4](#).

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

[b] OpenSSL libraries are in <NDDSHOME>/third\_party/openssl-3.5.1/<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
x64QNX8.0qcc_cxx12.2.0	Static Release	-Vgcc/12.2.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -O -DNDEBUG -fPIC -fvisibility=hidden
	Static Debug	-Vgcc/12.2.0,gcc_ntox86_64 -lang-c++ -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O0 -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC
	Dynamic Release	-Vgcc/12.2.0,gcc_ntox86_64 -lang-c++ -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -O -DNDEBUG -fPIE -fvisibility=hidden
	Dynamic Debug	-Vgcc/12.2.0,gcc_ntox86_64 -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O0 -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -g -fPIC
armv8QNX8.0qcc_cxx12.2.0	Static Release	-Vgcc/12.2.0,gcc_ntoaarch64le -lang-c++ -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -O -DNDEBUG -fPIC -fvisibility=hidden
	Static Debug	-Vgcc/12.2.0,gcc_ntoaarch64le -lang-c++ -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O0 -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -g -fPIC
	Dynamic Release	-Vgcc/12.2.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -O -DNDEBUG -fPIC -fvisibility=hidden
	Dynamic Debug	-Vgcc/12.2.0,gcc_ntoaarch64le -DFD_SETSIZE=512 -DSIZEOF_LONG=8 -O0 -Y_cxx -fexceptions -fno-strict-aliasing -fno-semantic-interposition -O0 -g -fPIC

## 7.3 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]

## 7.4 Thread Configuration

### 7.4.1 Thread Settings and Thread-Priority Definitions

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

[a]  $\{\text{NDDSHOME}\}$  represents the root directory of your *Connex* installation.  $\{\text{LD\_LIBRARY\_PATH}\}$  represents the value of the LD\_LIBRARY\_PATH variable prior to changing it to support *Connex*.

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

## 7.4.2 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.

### 7.4.3 Automatic Thread-Specific Storage Cleanup

QNX systems support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore not required to clean up thread-specific storage in user threads (although it is safe to do so).

## 7.5 Socket Buffer Size Configuration

See [Table 7.13 UDP send\\_socket\\_buffer\\_size](#) and [Table 7.14 UDP receive\\_socket\\_buffer\\_size](#). For more information on the **send\_socket\_buffer\_size** and **receive\_socket\_buffer\_size** properties, see [Setting Builtin Transport Properties with the PropertyQosPolicy, in the RTI Connex Core Libraries User's Manual](#).

**Table 7.13 UDP send\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size <b>net.inet.udp.sendspace</b>
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>kern.ipc.maxsockbuf</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size <b>net.inet.udp.sendspace</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>kern.ipc.maxsockbuf</b>

**Table 7.14 UDP receive\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size <b>net.inet.udp.recvspace</b>
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>kern.ipc.maxsockbuf</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size <b>net.inet.udp.recvspace</b>
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Uses maximum send socket buffer size <b>kern.ipc.maxsockbuf</b>

## 7.6 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

## 8.1 Building Applications for VxWorks Platforms

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

- Compiling a *Connex*t application for VxWorks depends on the development platform. For more information, such as specific compiler flags, see the *VxWorks Programmer's Guide*.
- 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 `VSBDIR` that points to the VSB. In the generated makefile, find this line and change it to match your VSB directory:

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

**Note:** RTI uses a VSB based on the `itl_generic` BSP provided by Wind River to build the *Connex*t 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*t 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*t 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*t libraries, and another static DKM containing the subscriber application + *Connex*t libraries. When those two modules are loaded into the kernel, all the *Connex*t 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*t 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*t libraries (in particular, *libniddsc*) 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 4](#).

[Table 8.1 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 88](#).

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.

#### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*t, *debug* libraries are created with debug symbols to facilitate debugging with **gdb**, for example. *Release* libraries do not contain debug information.

The *Connex*t debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production

environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

### Additional Documentation:

See the [RTI Connex Core Libraries Getting Started Guide Addendum for Embedded Systems](#).

**Table 8.1 Building Instructions for VxWorks Architectures**

API	Library Format	Required RTI Libraries <sup>[a]</sup> , <sup>[b]</sup>	Required Kernel Components	Required Compiler Flags
C++ (Traditional and Modern APIs)	Static Release	libnndscorez.a libnndscz.a  libnndscppz.a or libnndscpp2z.a  librtconnextmsgcppz.a or librtconnextmsgcpp2z.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  librtconnextmsgcppzd.a or librtconnextmsgcpp2zd.a		
	Dynamic Release	libnndscore.so libnndsc.so  libnndscpp.so or libnndscpp2.so  librtconnextmsgcpp.so or librtconnextmsgcpp2.so		
	Dynamic Debug	libnndscored.so libnndscd.so  libnndscppd.so or libnndscpp2d.so  librtconnextmsgcppd.so or librtconnextmsgcpp2d.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 8.1 Building Instructions for VxWorks Architectures**

API	Library Format	Required RTI Libraries <sup>[a]</sup> , <sup>[b]</sup>	Required Kernel Components	Required Compiler Flags
C	Static Release	libnbdscorz.a libnbdscz.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	libnbdscorz.d.a libnbdsczd.a librticonnextmsgcz.d.a		
	Dynamic Release	libnbdscore.so libnbdsc.so librticonnextmsgc.so		
	Dynamic Debug	libnbdscore.d.so libnbdscd.so librticonnextmsgcd.so		

### 8.1.3 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to the *Supported Platforms* section of the [RTI Connex Core Libraries Release Notes](#) to see if these libraries are supported on your platform.

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

### 8.1.3.1 Libraries Required for Distributed Logger

[Table 8.2 Additional Libraries for using RTI Distributed Logger](#) lists the additional libraries you will need in order to use *Distributed Logger*.

**Table 8.2 Additional Libraries for using RTI Distributed Logger**

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

### 8.1.3.2 Libraries Required for Monitoring

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 8.3 Additional Libraries for Monitoring](#) must appear *first* in the list of libraries to be linked.

**Table 8.3 Additional Libraries for Monitoring**

Library Format	Monitoring Libraries <sup>[c]</sup>
Dynamic Release	librtimonitoring.so <sup>[d]</sup>
Dynamic Debug	librtimonitoringd.so <sup>[e]</sup>

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

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

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

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

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

**Table 8.3 Additional Libraries for Monitoring**

Library Format	Monitoring Libraries <sup>[a]</sup>
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 8.4 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 8.4 Additional Libraries for Using Real-Time WAN Transport APIs**

Library Format	Real-Time WAN Transport Libraries <sup>[b]</sup>
Dynamic Release	libnddsrtw.so
Dynamic Debug	libnddsrtwd.so
Static Release	libnddsrtwz.a
Static Debug	libnddsrtwzd.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 8.5 Additional Libraries for Zero Copy Transfer Over Shared Memory](#).

**Table 8.5 Additional Libraries for Zero Copy Transfer Over Shared Memory**

Library Format	Zero Copy Over Shared Memory Libraries <sup>[c]</sup>
Dynamic Release	libnddsmetp.so
Dynamic Debug	libnddsmetpd.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>.

**Table 8.5 Additional Libraries for Zero Copy Transfer Over Shared Memory**

Library Format	Zero Copy Over Shared Memory Libraries <sup>[a]</sup>
Static Release	libnddsmetpz.a
Static Debug	libnddsmetpzd.a

## 8.2 Running User Applications

Table 8.6 Running Instructions for VxWorks Architectures below provides details on the environment variables that must be set at runtime for a VxWorks architecture.

**Table 8.6 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>" <sup>[b]</sup>
	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>.

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

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

## 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`.

## 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 [Chapter 8 VxWorks Platforms on page 85](#) 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*t applications are restarted, there should be no problem.

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

All *Connex*t 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 Transports

### 8.8.1 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.

### 8.8.2 How To Run Connex Libraries in Kernels Built without Shared Memory

Since *Connex*t 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*t 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.

## 8.9 Unsupported Features

These features are not supported on VxWorks platforms:

- Backtrace
- Controlling CPU Core Affinity
- Durable Writer History and Durable Reader State
- 'Find Package' CMake script

Please refer to the *Supported Platforms* section of the [RTI Connex Core Libraries Release Notes](#) to see which APIs, transports, services, and other features are supported for each platform.

## 8.10 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 VxWorks platforms.

## 8.11 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.

## 8.12 Thread Configuration

### 8.12.1 Thread Settings, Thread-Priority Definitions, and Thread Kinds

See these tables:

- [Table 8.7 Thread Setting for VxWorks Platforms on the next page](#)
- [Table 8.8 Thread-Priority Definitions for VxWorks Platforms on the next page](#)

- [Table 8.9 Thread Kinds for VxWorks Platforms on the next page](#)

**Table 8.7 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	40 * 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	40 * 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 * 40 * 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 * 40 * 1024
	cpu_list	CPU core affinity not supported
	cpu_rotation	CPU core affinity not supported

**Table 8.8 Thread-Priority Definitions for VxWorks Platforms**

Thread-Priority Definition	Operating-System Priority
THREAD_PRIORITY_DEFAULT	100
THREAD_PRIORITY_HIGH	68

**Table 8.8 Thread-Priority Definitions for VxWorks Platforms**

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

**Table 8.9 Thread Kinds for VxWorks Platforms**

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

## 8.12.2 Automatic Thread-Specific Storage Cleanup

VxWorks systems do not support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore required to clean up thread-specific storage in user threads.

## 8.13 Socket Buffer Size Configuration

See [Table 8.10 UDP send\\_socket\\_buffer\\_size](#) and [Table 8.11 UDP receive\\_socket\\_buffer\\_size](#). For more information on the **send\_socket\_buffer\_size** and **receive\_socket\_buffer\_size** properties, see [Setting Builtin Transport Properties with the PropertyQosPolicy, in the RTI Connex Core Libraries User's Manual](#).

[a] See VxWorks manuals for more information.

**Table 8.10 UDP send\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT will be used)
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT will be used)

**Table 8.11 UDP receive\_socket\_buffer\_size**

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT will be used)
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT will be used)

# Chapter 9 Windows Platforms

## 9.1 Summary of Windows Platforms and Supported Features

The following table shows the supported products/features for each architecture. A ● means supported in this release, an empty cell means unsupported in this release.

**Table 9.1 Windows Systems - Supported Features**

RTI Architecture [a]	Supported Features						
Languages							
	Java [b]	Python [c]	.NET (C#) [d]	C/C++	Modern C++ [e] , [f]		
Windows® (Intel®) Windows 11; Windows Server 2022 x64Win64VS2017	●	● [g]	●	●	● [h]		

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Built with Eclipse Temurin OpenJDK 21.0.7, compatible with Java 8 and above. Also tested with AdoptOpenJDK 11.0.13 and 17.0.6.

[c] Tested with Python 3.8-3.12.

[d] Built with .NET 6, compatible with .NET 5+ (and newer). Also supported on .NET Core 2.1 (and above) and .NET Framework 4.8.3 (and above).

[e] Tested with C++11 unless stated otherwise.

[f] Supporting Modern C++ means supporting the RPC feature unless otherwise stated.

[g] Make sure to install 64-bit Python, not 32-bit Python, which may be installed by default even on 64-bit Windows.

[h] Tested with C++14.

## 9.1 Summary of Windows Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
<b>Transports</b>							
	UDP/IPv4	UDP/IPv6	Multicast	TCP/IPv4	Shared Memory (w/ zero copy) <sup>[b]</sup>		
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•	• <sup>[c]</sup>	•	•	•		
<b>Infrastructure Services</b>							
	Persistence Service <sup>[d]</sup>	Routing Service	Recording and Replay Services	Web Integration Service			
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•	•	•	•			
<b>Connex Professional Libraries</b>							
	Request/Reply	Monitoring Library	Distributed Logger	LBED <sup>[e]</sup>	Monitoring Library 2.0		
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•	•	•	•	See "Observability Framework" rows below		

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> Zero-copy transfer over shared memory is NOT supported for Java, Ada, .NET and Python programming languages. It is only supported for C, Traditional C++, and Modern C++ programming languages.

<sup>[c]</sup> No Traffic Class support.

<sup>[d]</sup> Tested with filesystem only in PERSISTENT mode.

<sup>[e]</sup> Supports dynamic linking only.

## 9.1 Summary of Windows Platforms and Supported Features

RTI Architecture <sup>[a]</sup>	Supported Features						
Core Library Features							
	Monotonic Clock	CPU Core Affinity	Durable writer history & reader state	RTI DDS thread name	Backtrace <sup>[b]</sup>	Cmake Find Package	
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•		•	• <sup>[c]</sup>	•	•	
Tools							
	Shapes Demo	Launcher	Monitor	Admin Console	System Designer	rtiddsgen server	Utilities (rtiddsping, rtiddsspy)
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•	•	•	•	• <sup>[d]</sup>	•	•
Observability Framework							
	Observability Collector Service	Monitoring Library 2.0					
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017		•					

<sup>[a]</sup> Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

<sup>[b]</sup> If you want backtrace information: on all Linux architectures, compile with `-fno-omit-frame-pointer` and add linker flag `-rdynamic`. For Linux architectures on Arm CPUs, also compile with `“-funwind-tables”`. On all Windows architectures, you need the `Dbghelp.dll` and `Ntdll.dll` libraries.

<sup>[c]</sup> NOT supported in release mode.

<sup>[d]</sup> Tested on x64 Windows, with Chrome version 112 and Firefox version 108.

RTI Architecture <sup>[a]</sup>	Supported Features						
Security Extensions							
	Security Plugins (for OpenSSL) <sup>[b]</sup>	Security Plugins (for wolfSSL) <sup>[c]</sup>	TLS Support <sup>[d]</sup>	Security Plugins SDK <sup>[e], [f]</sup>			
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•		•	•			
Add-ons							
	Cloud Discovery Service	Real-Time WAN Transport	Ada Language Support	Limited Bandwidth Plugins	Queuing Service (Experimental)		
Windows (Intel) Windows 11; Windows Server 2022 x64Win64VS2017	•	•		•	•		

## 9.1.1 Monotonic Clock Support

See [Configuring the Clock per DomainParticipant, in the RTI Connex Core Libraries User's Manual](#) for information on the monotonic clock.

## 9.1.2 Support for 'Find Package' CMake Script

For information on using the 'Find Package' CMake script, see [2.5 Building with CMake on page 8](#).

## 9.1.3 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

[a] Supports DDS 1.4, RTPS 2.5, and DDS X-Types 1.3.

[b] Tested with OpenSSL3.5.1 unless stated otherwise.

[c] Tested with WolfSSL 5.5.1.

[d] Tested with OpenSSL3.5.1 unless stated otherwise.

[e] Tested with OpenSSL3.5.1 unless stated otherwise.

[f] Tested with WolfSSL 5.5.1.

“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.

See also [Logging a Backtrace for Failures, in the RTI Connex Core Libraries User's Manual](#).

## 9.2 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 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 4](#)

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.

### Release and Debug Terminology:

Both *release* and *debug* versions of the libraries are provided. For *Connex*, *debug* libraries are created with debug symbols to facilitate debugging with **gdb**, for example. *Release* libraries do not contain debug information.

The *Connex* debug libraries are intended solely for development and debugging purposes. These libraries are not optimized for performance or resource consumption and may include additional diagnostic information that can affect runtime behavior. They are not intended for use in production environments. Please ensure that only the release libraries are used in production deployments. Debug libraries can be identified by the suffix "d" in their names.

## 9.3 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 4](#)
2. Add the path to your *Connex* installation folder to an 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**
    - **libcmt**
6. If you want to use the MFC library in your 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 on the next page](#), depending on the version of the libraries you plan to link against.

Table 9.2 Building Instructions for Windows Architectures

API	Library Format	Required RTI Libraries or Jar Files <sup>[a]</sup> , <sup>[b]</sup>	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	Required RTI Libraries or Jar Files <sup>[a], [b]</sup>	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		

### 9.3.1 Additional Libraries for Other Features

This section discusses libraries for features that are not supported for every platform. Please refer to [Table 9.1 Windows Systems - Supported Features](#) to see if these libraries are supported on your platform.

[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`.

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.

### 9.3.1.1 Libraries Required for Distributed Logger

[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 <sup>[a]</sup>		Dynamic <sup>[b]</sup>	
	Release	Debug	Release	Debug
C	rtidcz.lib	rtidczd.lib	rtidc.lib rtidc.dll	rtidcd.lib rtidcd.dll
C++ (Traditional API)	rtidcz.lib rtidcppz.lib	rtidczd.lib rtidcppzd.lib	rtidc.lib rtidc.dll rtidcpp.lib rtidcpp.dll	rtidcd.lib rtidcd.dll rtidcppd.lib rtidcppd.dll
Java	N/A	N/A	distlog.jar distlogdatamodel.jar	distlogd.jar distlogdatamodeld.jar

### 9.3.1.2 Libraries Required for Monitoring

To use the Monitoring APIs, reference the libraries in [Table 9.4 Additional Libraries for Using Monitoring](#).

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.

[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 <sup>[a]</sup>
Dynamic Release	rtimonitoring.lib rtimonitoring.dll
Dynamic Debug	rtimonitoringd.lib rtimonitoringd.dll
Static Release	rtimonitoringz.lib Psapi.lib
Static Debug	rtimonitoringzd.lib Psapi.lib

### 9.3.1.3 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 [9.3.1 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 <sup>[b]</sup>
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 [9.3.3 Location of OpenSSL Libraries on page 110](#).

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

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

### 9.3.1.4 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 <sup>[a]</sup>
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 <sup>[b]</sup>	OpenSSL Libraries	System Libraries
Dynamic Release	nddstls.lib nddstls.dll	libssl.lib libssl-<version>.dll	(none)
Dynamic Debug	nddstlsd.lib nddstlsd.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 [9.3.3 Location of OpenSSL Libraries on page 110](#).

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

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

### 9.3.1.5 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 <sup>[a]</sup>
Dynamic Release	nddsmetp.lib nddsmetp.dll
Dynamic Debug	nddsmetpd.lib nddsmetpd.dll
Static Release	nddsmetpz.lib
Static Debug	nddsmetpzd.lib

## 9.3.2 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 [Chapter 2 Building Applications—Notes for All Platforms on page 4](#).

**Table 9.9 Library-Creation Details for Windows Architectures**

RTI Architecture	Library Format	Compiler Flags Used by RTI
x64Win64VS2017	Static Release	/O2 /Ob2 /D WIN32 /D _WINDOWS /D NDEBUG /D SIZEOF_LONG=8 /D _WIN32_WINNT=0x0600 /D _CRT_SECURE_NO_DEPRECATED /D WIN32_LEAN_AND_MEAN /D COMPILED_FROM_DSP /D STDC99 /D _MBCS /EHsc /MD /GS /fp:precise /Zc:wchar_t /Zc:forScope /Zc:inline /GR /std:c++14 /Gd /FC
	Static Debug	/Zi /Od /Ob0 /D WIN32 /D _WINDOWS /D _SCL_SECURE_NO_WARNINGS /D SIZEOF_LONG=8 /D _WIN32_WINNT=0x0600 /D WIN32_LEAN_AND_MEAN /D COMPILED_FROM_DSP /D STDC99 /D _MBCS /EHsc /RTC1 /MDd /GS /fp:precise /Zc:wchar_t /Zc:forScope /Zc:inline /GR /Gd /FC
	Dynamic Release	/O2 /Ob2 /GL /D WIN32 /D _WINDOWS /D _SCL_SECURE_NO_WARNINGS /D NDEBUG /D SIZEOF_LONG=8 /D _WIN32_WINNT=0x0600 /D WIN32_LEAN_AND_MEAN /D COMPILED_FROM_DSP /D _MBCS /EHsc /MD /GS /fp:precise /Zc:wchar_t /Zc:forScope /Zc:inline /GR /Gd
	Dynamic Debug	/Zi /Od /Ob0 /D WIN32 /D _WINDOWS /D SIZEOF_LONG=8 /D _WIN32_WINNT=0x0600 /D WIN32_LEAN_AND_MEAN /D COMPILED_FROM_DSP /D _MBCS /EHsc /RTC1 /MDd /GS /fp:precise /Zc:wchar_t /Zc:forScope /Zc:inline /GR /Gd /FC

[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
Windows architectures for Java	Dynamic Release	-target 1.8 -source 1.8
	Dynamic Debug	-target 1.8 -source 1.8 -g

### 9.3.3 Location of OpenSSL Libraries

The OpenSSL libraries are installed here:

- OpenSSL **.lib** files are in:  
`<NDDSHOME>\third_party\openssl-3.5.1\<architecture>\<format>\lib.`
- OpenSSL **.dll** files are in:  
`<NDDSHOME>\third_party\openssl-3.5.1\<architecture>\<format>\bin.`

Where:

- `<architecture>` is your architecture string, as listed in [Chapter 9 Windows Platforms on page 98](#), 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.

## 9.4 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 [Chapter 9 Windows Platforms on page 98](#).
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 on the next page](#).

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 Connext Getting Started Guide](#).

**Table 9.10 Running Instructions for Windows Architectures**

RTI Architecture	Library Format	Environment Variables <sup>[a]</sup>
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.

## 9.5 Thread Configuration

### 9.5.1 Thread Settings, Thread-Priority Definitions, and Thread Kinds

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

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

**Table 9.11 Thread Settings for Windows Platforms**

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

**Table 9.12 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 <sup>[a]</sup>
DDS_THREAD_SETTINGS_FLOATING_POINT	N/A
DDS_THREAD_SETTINGS_STUDIO	
DDS_THREAD_SETTINGS_REALTIME_PRIORITY	
DDS_THREAD_SETTINGS_PRIORITY_ENFORCE	

## 9.5.2 Automatic Thread-Specific Storage Cleanup

Windows systems support automatic thread-specific storage cleanup. Use of the **DomainParticipantFactory::unregister\_thread** API is therefore not required to clean up thread-specific storage in user threads (although it is safe to do so).

## 9.6 Socket Buffer Size Configuration

See [Table 9.14 UDP send\\_socket\\_buffer\\_size](#) and [Table 9.15 UDP receive\\_socket\\_buffer\\_size](#). For more information on the **send\_socket\_buffer\_size** and **receive\_socket\_buffer\_size** properties, see [Setting Builtin Transport Properties with the PropertyQosPolicy, in the RTI Connex Core Libraries User's Manual](#).

[a] See Windows manuals for additional information.

Table 9.14 UDP send\_socket\_buffer\_size

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV4_SEND_SOCKET_BUFFER_SIZE_DEFAULT will be used)
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default send socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV6_SEND_SOCKET_BUFFER_SIZE_DEFAULT will be used)

Table 9.15 UDP receive\_socket\_buffer\_size

Property	Behavior
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size
NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV4_RECV_SOCKET_BUFFER_SIZE_DEFAULT will be used)
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT	Uses default receive socket buffer size
NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX	Not supported (NDDS_TRANSPORT_UDPV6_RECV_SOCKET_BUFFER_SIZE_DEFAULT will be used)

## 9.7 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.
```

## 9.8 Requirements when Using Microsoft Visual Studio Code

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

## 9.8.1 When Using Visual Studio 2017 – Redistributable Package Requirement

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

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

## 9.8.2 When Using Visual Studio 2019 – Redistributable Package Requirement

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

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

## 9.8.3 When Using Visual Studio 2022 – Redistributable Package Requirement

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

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