# RTI Limited Bandwidth Plugins

## User's Manual

## Version 7.0.0

Printed in U.S.A. First printing.
September 2022.

## Trademarks



## Copy and Use Restrictions



## Notices

*Deprecations and Removals*

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

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

*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 sublicenses), 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."

**Technical Support**

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

# Contents

**Chapter 1 Introduction**

1.1 Provided Examples ..... 2

1.2 What is a Transport Plugin? ..... 2

1.3 What is Discovery? ..... 2

1.4 What is Limited Bandwidth Discovery? ..... 3

1.5 Configuring Transports with the Property QoS Policy in XML ..... 3

**Chapter 2 Paths Mentioned in Documentation ..... 5**

**Chapter 3 Limited Bandwidth Participant Discovery Plugin**

3.1 Creating the LBPD Plugin Configuration File ..... 7

3.2 Configuring the LBPD Plugin in Connext ..... 10

3.3 Optimizing the Plugin ..... 12

3.3.1 Initial Announcements ..... 13

3.3.2 Liveliness ..... 13

**Chapter 4 Limited Bandwidth Endpoint Discovery Plugin**

4.1 Generate Connext Project ..... 16

4.2 Add Name to Each DomainParticipant ..... 18

4.3 Add an RTPS Object ID to Each Endpoint ..... 20

4.4 Enable LBED Plugin on Each DomainParticipant ..... 21

4.4.1 Using the Builtin Discovery Plugins Mask (Recommended) ..... 21

4.4.2 Using the LBED Plugin Properties ..... 22

4.5 Add the DDS-XML Definition of the Entities ..... 23

4.5.1 Using the USER_QOS_PROFILES.xml ..... 24

4.5.2 Using a Separate XML File ..... 31

4.6 Run the Applications ..... 35

4.6.1 Configure the Environment in Both Command Prompts ..... 35

4.6.2 Check Communication ..... 35

4.6.3 Check that Your Endpoints Discover Each Other Statically ..... 37
4.7 Troubleshooting ..... 38
4.7.1 System does not Recognize or Find rtiddsgen Command ..... 38
4.7.2 rtiddsgen Displays Error (Usually in Windows systems) ..... 39
4.7.3 rtiddsgen Gives Warnings ..... 39
4.7.4 Running Publisher/Subscriber Produces Errors ..... 39
4.7.5 Communication does not Occur between Publisher/Subscriber ([data: …] Messages not Displayed in Subscriber Prompt) ..... 40
4.8 Limitations ..... 40
4.9 LBED Properties ..... 41
4.10 Supported QoS ..... 43
4.11 Known Issues ..... 46
4.11.1 LBED not able to automatically load USER_QOS_PROFILES.xml if utilized API uses QosProvider and the file is not placed in default locations ..... 46
**Chapter 5 Limited Bandwidth RTPS Transport Plugin**
5.1 Understanding the RTPS Message Header ..... 48
5.1.1 Submessage Structure ..... 50
5.2 Configuring the LBRTPS Transport ..... 51
5.2.1 Configuring the LBRTPS Transport Plugin's 'Subtransport' Property ..... 56
5.2.2 Configuring the LBRTPS Transport Plugin's 'reduce_guidPrefix' Property ..... 59
**Chapter 6 Compression Real-Time Publish Subscribe Transport Plugin**
6.1 Transport-Related Limitations ..... 62
6.2 Configuring the ZRTPS Transport ..... 62
6.2.1 Configuring the ZRTPS Transport Plugin's 'Subtransport' Property ..... 67
6.2.2 Configuring the External Compression Library ..... 68

# Chapter 1 Introduction

The *RTI* ® *Limited Bandwidth Plugins* package includes:

- Discovery Plugins
  - Limited Bandwidth Participant Discovery Plugin

    Reduces discovery time and network traffic by obtaining some of the information about the participants from an XML file instead of from the normal discovery process, which requires all information to be sent dynamically over the network. All the participants must be known ahead of time and described in an XML file.
  - Limited Bandwidth Endpoint Discovery Plugin

    Reduces discovery time and network traffic by obtaining information about the endpoints from an XML file instead of from the normal discovery process, which requires the information to be sent dynamically over the network. All the endpoints must be known ahead of time and described in an XML file.
- Transport Plugins
  - Limited Bandwidth RTPS Transport Plugin

    Reduces the size of the message headers in the Real-Time Publish Subscribe (RTPS) packages sent over the network by the *RTI Connext*® software. The message headers are reduced by eliminating some fields and making other fields smaller.
  - Compression Real-Time Publish Subscribe Transport Plugin

    Compresses the RTPS packages sent over the network by *Connext*. You can configure how the packages are compressed, and even provide your own compression algorithm.

You can combine the abilities of these plugins or use them independently. When using more than one of the plugins, their properties must appear in the XML file in the order in which they should be executed.

## 1.1 Provided Examples

Examples are provided in these directories:

- **<path to examples>/connext_dds/c/limited_bandwidth_plugins**:
  - lbpdiscovery
  - lbrtps
  - zrtps
- **<path to examples>/connext_dds/c++11/limited_bandwidth_plugins**:
  - dil-stacking
  - lbediscovery

## 1.2 What is a Transport Plugin?

*Connext* sends data over a variety of transport networks. *Connext* has pluggable transport architecture. The core of *Connext* is transport agnostic, it does not make any assumptions about the actual transports used to send and receive messages.

*Connext* comes with standard UDPv4/IP and UDPv6/IP pluggable transports, as well as a shared memory transport; these transports are enabled by default. *Connext* also give you the ability to define new transport plugins and run utilizing them.

## 1.3 What is Discovery?

Discovery is the behind-the-scenes way in which *Connext* objects (*DomainParticipants*, *DataWriters*, and *DataReaders*) find out about each other. Each *DomainParticipant* maintains a database of information about all the active *DataReaders* and *DataWriters* in the same domain. This database is what makes it possible for *DataWriters* and *DataReaders* to communicate. To create and refresh the database, each application follows a common discovery process.

The default discovery mechanism in *Connext* is the one described in the DDS specification and is known as Simple Discovery Protocol, which includes two phases: Simple Participant Discovery and Simple Endpoint Discovery. The goal of these two phases is to build, for each *DomainParticipant*, a complete picture of all the entities that belong to the remote participants in its peers list, which is a list of nodes with which a participant may communicate.

During the Simple Participant Discovery phase, *DomainParticipants* learn about each other. The *DomainParticipant's* details are communicated to all other *DomainParticipants* in the same domain by sending participant declaration messages, also known as participant DATA submessages or participant announcements.

During the Simple Endpoint Discovery phase, *Connext* matches *DataWriters* and *DataReaders*. Information about each application's *DataReaders* and *DataWriters* is exchanged by sending publication/subscription declarations in DATA submessages (participant announcements), which we will refer to as publication DATAs and subscription DATAs. The Simple Endpoint Discovery phase uses reliable communication.

## 1.4 What is Limited Bandwidth Discovery?

The Limited Bandwidth Discovery Protocol reduces the amount of information exchanged between applications. This mechanism includes two phases: Limited Bandwidth Participant Discovery and Limited Bandwidth Endpoint Discovery. Both phases can be used separately from each other. Reducing traffic on the network reduces the discovery time.

Limited Bandwidth Discovery = Limited Bandwidth Participant Discovery
+ Limited Bandwidth Endpoint Discovery

## 1.5 Configuring Transports with the Property QoS Policy in XML

*Connext* provides a mechanism to dynamically load an external transport from an XML QoS profile, like the file generated by rtiddsgen **(USER_QOS_PROFILES.xml**). The Property QoS policy is used to achieve this purpose.

The Property QoS policy stores name/value (string) pairs that can be used to configure certain parameters of *Connext* that are not exposed through formal QoS policies. *Connext* uses this mechanism to configure external transports.

**Syntax for Setting the Property QoS Policy in an XML QoS profile:**

```
<qos_library name="Property_Library">
    <qos_profile name="Property_Profile">
        <domain_participant_qos>
            ...
            <property>
                <value>
                    <element>
                        <name>Property1</name>
                        <value>example</value>
                    </element>
                    <element>
                        <name>Property2</name>
                        <value>example</value>
                    </element>
```

```
                    ...
                </value>
            </property>
            ...
        </domain_participant_qos>
    </qos_profile>
</qos_library>
```

For more general information, see *Configuring QoS with XML*, in the RTI Connext Core Libraries User's Manual.

For specific information on setting properties for each of the *Limited Bandwidth Plugins*, see their respective chapters in this document.

# Chapter 2 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

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

  - macOS® systems:
    **/Applications/rti_connext_dds-7.0.0**
  - Linux systems, non-*root* user:
    **/home/<*your user name*>/rti_connext_dds-7.0.0**
  - Linux systems, *root* user:
    **/opt/rti_connext_dds-7.0.0**
  - Windows® systems, user without Administrator privileges:
    **<*your home directory*>\rti_connext_dds-7.0.0**
  - Windows systems, user with Administrator privileges:
    **C:\Program Files\rti_connext_dds-7.0.0**

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

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

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

```
"C:\Program Files\rti_connext_dds-7.0.0\bin\rtiddsgen"
```

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

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

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

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

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

  Default path to the examples:

  - macOS systems: **/Users/<*your user name*>/rti_workspace/7.0.0/examples**
  - Linux systems: **/home/<*your user name*>/rti_workspace/7.0.0/examples**
  - Windows systems: ***<your Windows documents folder>*\rti_workspace\7.0.0\examples**

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

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

# Chapter 3 Limited Bandwidth Participant Discovery Plugin

Limited Bandwidth Participant Discovery (LBPD) is achieved with a file-based plugin. Part of the information about the participants is obtained from an XML file instead of being sent dynamically over the network. This method can reduce discovery time and reduce traffic on the network. However, for LBPD to work, all the participants must be known ahead of time and described in an XML file.

The LBPD plugin reduces, but does not eliminate, the network traffic required to exchange participant information. It does this by allowing you to define some of the remote participant data, such as the product version and the RTPS protocol version, in an XML file.

The correlation between the remote participant information defined in the XML file and the information received on the network is done using the RTPS participant identifier (key) first and the participant name second if the identifier is not defined in the XML file (see Table 3.1 Configuration Options for LBPD Plugin for additional details).

This chapter describes how to configure the *RTI Limited Bandwidth Participant Discovery Plugin* and set up your *Connext* application to use the plugin.

You will need two XML files, one for the discovery plugin (see 3.1 Creating the LBPD Plugin Configuration File below) and one for *Connext* (3.2 Configuring the LBPD Plugin in Connext on page 10).

> You must link with the *dynamic* version of the *Connext* libraries. See the RTI Connext Core Libraries Platform Notes for details.

## 3.1 Creating the LBPD Plugin Configuration File

To use LBPD, you need an XML file that describes all the remote participants. These remote participants must be configured exactly the same as their original QoS properties.

You will specify the name of this file when you configure the plugin in the QoS Profiles XML file (**USER_QOS_PROFILES.xml**) described in 3.2 Configuring the LBPD Plugin in Connext on page 10; see dds.discovery.participant.<string>.config_file

The main structure of this file is:

```
<LBPDiscoveryPluginProfile>
    <participant name="Participant1">
    ...
    </participant>
    <participant name=”Participant2">
    ...
    </participant>
</LBPDiscoveryPluginProfile>
```

Let’s look at an example of this file (you can find it in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbpdiscovery/LBPDiscoveryPluginExamplePublisher.xml**.

```
<LBPDiscoveryPluginProfile>
    <participant name="Publisher">
        <key>
            <rtps_host_id>RTPS_AUTO_ID</rtps_host_id>
            <rtps_app_id>RTPS_AUTO_ID</rtps_app_id>
            <rtps_instance_id>RTPS_AUTO_ID</rtps_instance_id>
        </key>
        <rtps_protocol_version>
            <major>2</major>
            <minor>1</minor>
        </rtps_protocol_version>
        <rtps_vendor_id>
            <vendorId>1,1</vendorId>
        </rtps_vendor_id>
        <product_version>
            <major>6</major>
            <minor>x</minor>
            <release>y</release>
            <revision>z</revision>
        </product_version>
    </participant>
</LBPDiscoveryPluginProfile>
```

(In <product_version>, *x*, *y* and *z* represent the number of the current release.)

The supported participant configuration options are described in the following tables. They are all optional.

Some descriptions also point out related *Connext* documentation. For example, “See documentation on the Entity QoS policy” means you should see that section in the RTI Connext Core Libraries User's Manual or the *Connext* API Reference HTML documentation**.**

## Table 3.1 Configuration Options for LBPD Plugin

| Option Name | Option Value and Description |
|---|---|
| key | The RTPS identifier of the participant. See documentation on the WireProtocol QoS policy. If a key is not set or the values are set to RTPS_AUTO_ID, the correlation between the participant information defined in the XML file and the information received from the network will be done using the participant's entity name.<br>Example:<br>`<key>`<br>`  <rtps_host_id>123</rtps_host_id>`<br>`  <rtps_app_id>255</rtps_app_id>`<br>`  <rtps_instance_id>348</rtps_instance_id>`<br>`</key>` |
| liveliness_lease_ duration | The liveliness lease duration for the participant.<br>Schema:<br>`<liveliness_lease_duration>`<br>`  <sec>[number \| DURATION_ZERO_SEC \|`<br>`  DURATION_INFINITE_SEC]`<br>`  </sec>`<br>`  <nanosec>[number \| DURATION_ZERO_NSEC \|`<br>`  DURATION_INFINITE_NSEC]`<br>`  </nanosec>`<br>`</liveliness_lease_duration>`<br>Example:<br>`<liveliness_lease_duration>`<br>`  <sec>100</sec>`<br>`  <nanosec>DURATION_ZERO_NSEC</nanosec>`<br>`</liveliness_lease_duration>` |
| rtps_protocol_ version | The version number of the RTPS protocol being used. See documentation on the ParticipantBuiltinTopicData structure.<br>Example:<br>`<rtps_protocol_version>`<br>`  <major>2</major>`<br>`  <minor>1</minor>`<br>`</rtps_protocol_version>` |
| rtps_vendor_id | The identifier of the RTPS vendor. See documentation on ParticipantBuiltinTopicData. RTI's identifier is 1,1.<br>Example:<br>`<rtps_vendor_id><vendorId>1,1</vendorId><rtps_vendor_id>` |
| participant_name | The name of the remote participant, as set in EntityName QoS policy.<br>Example:<br>`<participant name="Participant1"/>`<br>The participant name is used to correlate the information defined in the XML file with the information received on the network in the absence of the key property. If both the key property and the participant name are not defined, the discovery plugin will report the following error:<br>LBPDiscoveryPluginTypeReaderListenerOnDataAvailable:<br>Cannot find the participant in the database |
| product_version | The version number for the plugin.<br>Example (where *x*, *y*, and *z* represent numbers of the current release):<br>`<product_version>`<br>`  <major>6</major>`<br>`  <minor>x</minor>`<br>`  <release>y</release>`<br>`  <revision>z</revision>`<br>`</product_version>` |

## 3.2 Configuring the LBPD Plugin in Connext

This section describes how to configure the properties for the LBPD plugin in the XML QoS Profile file used by *Connext* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQosPolicy for your *Connext* application's *DomainParticipant*. (See the "PROPERTY QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual.)

Let's look at an example XML file, which you can find in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbpdiscovery/USER_QOS_PROFILES.xml**:

```
<domain_participant_qos>
    ...
    <property>
        <value>
            <!-- Specify the library -->
            <element>
                <name>dds.discovery.participant.lbpdiscovery.library</name>
                <value>rtilbpdisc</value>
            </element>
            <!-- Specify the creation function -->
            <element>
                <name>
                    dds.discovery.participant.lbpdiscovery.create_function
                </name>
                <value>DDS_LBPDiscoveryPlugin_create</value>
            </element>
            <!-- Specify the discovery configuration file.
                 Change this property to use your own file. -->
            <element>
                <name>dds.discovery.participant.lbpdiscovery.config_file</name>
                <value>LBPDiscoveryPluginExampleSubscriber.xml</value>
            </element>
            <!-- Load LBP Participant Discovery plugin -->
            <element>
                <name>dds.discovery.participant.load_plugins</name>
                <value>dds.discovery.participant.lbpdiscovery</value>
            </element>
            <!-- Specify the verbosity -->
            <element>
                <name>dds.discovery.participant.lbpdiscovery.verbosity</name>
                <value>0</value>
             </element>
        </value>
    </property>
    ...
</domain_participant_qos>
```

Table 3.2 LBPD Configuration Properties for Connext describes the name/value pairs that you can use to configure the LBPD plugin.

## Table 3.2 LBPD Configuration Properties for Connext

| Property Name | Property Value and Description |
|---|---|
| dds.discovery.participant.<br>load_plugins | Required.<br>String indicating the prefix name of the plugin that will be loaded by *Connext*.<br>Set the value to **dds.discovery.participant.<string>**, where <string> can be any string you want, as long as you use the same string consistently for all the properties in this table. Our example uses lbpdiscovery:<br>`<element>`<br>`<name>dds.discovery.participant.load_plugins</name>`<br>`<value>dds.discovery.participant.lbpdiscovery</value>`<br>`</element>` |
| dds.discovery.participant.<br><*string*>.library | Required.<br>The name of the dynamic library that contains the LBPD plugin implementation. This library must be in the path during run time for use by *Connext*.<br>Set the value to **rtilbpdisc**.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.discovery.participant.lbpdiscovery.library`<br>`</name>`<br>`<value>rtilbpdisc</value>`<br>`</element>` |
| dds.discovery.participant.<br><*string*>.create_function | Required.<br>The name of the function that will be called by *Connext* to create an instance of the LBPD plugin**.**<br>Set the value to **DDS_LBPDiscoveryPlugin_create**.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.discovery.participant.lbpdiscovery.create_function`<br>`</name>`<br>`<value>DDS_LBPDiscoveryPlugin_create</value>`<br>`</element>` |
| dds.discovery.participant.<br><*string*>.config_file | Required.<br>The name of the discovery configuration file, described in 3.1 Creating the LBPD Plugin Configuration File on page 7.<br>Set the value to the name of your own file.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.discovery.participant.lbpdiscovery.config_file`<br>`</name>`<br>`<value>LBPDiscoveryPluginExampleSubscriber.xml</value>`<br>`</element>` |

**Table 3.2 LBPD Configuration Properties for Connext**

| Property Name | Property Value and Description |
|---|---|
| dds.discovery.participant.<*string*>.verbosity | Optional.<br>The verbosity for the plugin, for debugging purposes.<br>• -1: Silent<br>• 0: Exceptions only (default)<br>• 1: Warnings<br>• 2 and up: Debug<br>Example:<br>`<element>`<br>`<name>`<br>`dds.discovery.participant.lbpdiscovery.verbosity`<br>`</name>`<br>`<value>0</value>`<br>`</element>`<br>Note: the LBPD logging verbosity is per application. The last *DomainParticipant* using LBPD and **explicitly setting this property** will apply that setting to all the *DomainParticipants* using LBPD within the application. If not explicitly set, the verbosity will be left unchanged. Therefore, if no *DomainParticipant* has configured the LBPD verbosity, it will be left to the default value. |
| dds.discovery.participant.<*string*>.property_validation_action | Optional.<br>By default, property names given in the PropertyQoSPolicy are validated to avoid using incorrect or unknown names (for example, due to a typo). This property configures the validation of the property names associated with the plugin:<br>• VALIDATION_ACTION_EXCEPTION: validate the properties. Upon failure, log errors and fail.<br>• VALIDATION_ACTION_SKIP: skip validation.<br>• VALIDATION_ACTION_WARNING: validate the properties. Upon failure, log warnings and do not fail.<br>If this property is not set, the plugin property validation behavior will be the same as that of the *DomainParticipant*, which by default is VALIDATION_ACTION_EXCEPTION. See the "Property Validation" section in the RTI Connext Core Libraries User's Manual. |

In addition to setting the properties described above, the **builtin_discovery_plugins** mask (set in the DiscoveryConfigQosPolicy) should be set to SEDP. The default value of this mask is SDP (Simple Discovery Protocol). The SDP consists of two parts, Simple Participant Discovery Protocol (SPDP) and Simple Endpoint Discovery Protocol (SEDP). Using the LBPD plugin replaces the need for the SPDP, so the **builtin_discovery_plugins** should be set to SEDP. This tells *Connext* to only use the SEDP for endpoint discovery, since participant discovery will use the LBPD plugin. If you are using both the LBPD plugin and the LBED plugin, this mask should be set to MASK_NONE.

## 3.3 Optimizing the Plugin

You can reduce network bandwidth by changing some *Connext* properties in the file **USER_QOS_PROFILE xml**. For an example of this user profile, see the file utils/xml/USER_QOS_PROFILES.xml.

These optimizations apply to the LBPD and LBED plugins:

- 3.3.1 Initial Announcements below
- 3.3.2 Liveliness below


## 3.3.1 Initial Announcements

When a participant is enabled, by default it sends five announcements. You can reduce the number of initial announcements and the period between them with these properties:

- **initial_participant_announcements**
- **min_initial_participant_announcement_period**
- **max_initial_participant_announcement_period**

Example:

```
<domain_participant_qos>
    <discovery_config>
        <initial_participant_announcements>
            1
        </initial_participant_announcements>
        <min_initial_participant_announcement_period>
            <sec>1</sec>
            <nanosec>0</nanosec>
        </min_initial_participant_announcement_period>
        <max_initial_participant_announcement_period>
            <sec>1</sec>
            <nanosec>0</nanosec>
        </max_initial_participant_announcement_period>
    </discovery_config>
</domain_participant_qos>
```

## 3.3.2 Liveliness

The participant liveliness period can be increased with the property **participant_liveliness_assert_period**. If this property is increased, the property **participant_liveliness_lease_duration** must also be increased.

Example:

```
<domain_participant_qos>
    <discovery_config>
        <participant_liveliness_lease_duration>
            <sec>1000</sec>
            <nanosec>DURATION_ZERO_NSEC</nanosec>
        </participant_liveliness_lease_duration>
        <participant_liveliness_assert_period>
            <sec>300</sec>
            <nanosec>DURATION_ZERO_NSEC</nanosec>
        </participant_liveliness_assert_period>
    </discovery_config>
```

```
</domain_participant_qos>
```

# Chapter 4 Limited Bandwidth Endpoint Discovery Plugin

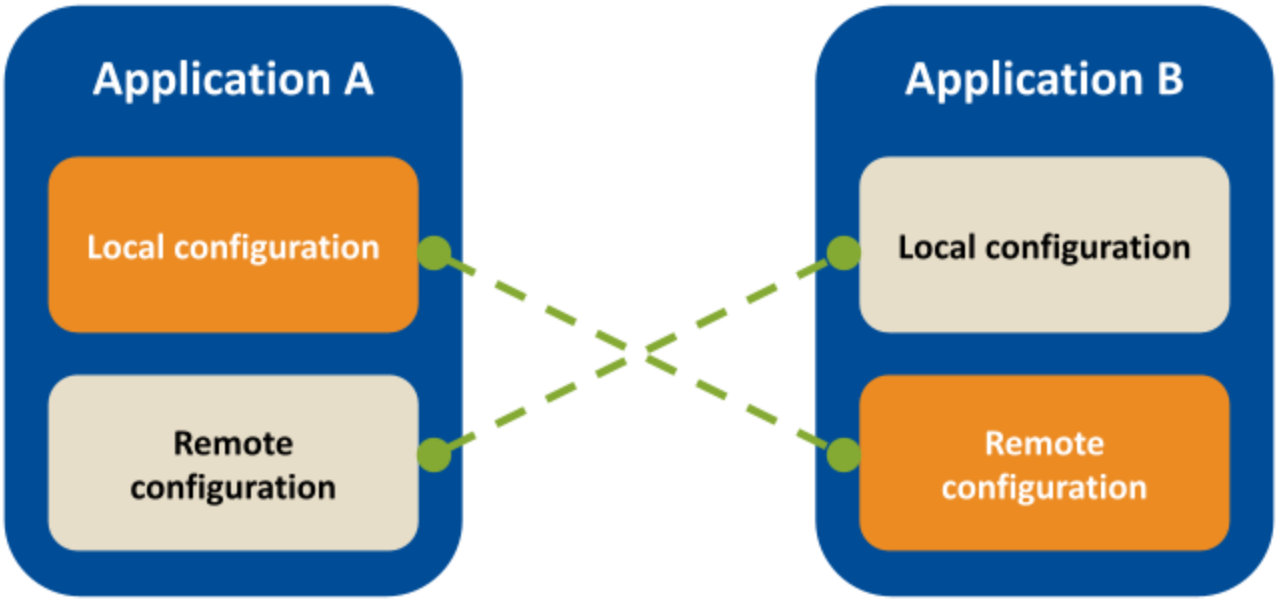
The *RTI® Limited Bandwidth Endpoint Discovery (LBED) Plugin* reduces discovery time and network traffic by locally defining information about the endpoints that need to be discovered in an XML file. The default dynamic discovery process, on the other hand, sends the information about the endpoints over the network. Therefore, LBED requires all the endpoints to be known ahead of time and each must be declared in an XML file.

*Limited Bandwidth Endpoint Discovery (LBED)* is achieved with a file-based plugin. Information about the endpoints is obtained from an XML file instead of being sent dynamically over the network. This method can reduce discovery time and network traffic. However, for LBED to work, all the endpoints must be known ahead of time and described in an XML file.

When using LBED in a system, each application has two different configuration kinds: the local configuration, which is the QoS policies, *Topics*, and types the application's endpoints use, and the remote configuration required by LBED, which is the QoS policies, *Topics*, and types from other applications' endpoints that the application needs to discover. Local configuration is usually specified using XML (e.g., **USER_QOS_PROFILES.xml**). Remote configuration is always specified using XML (i.e., **USER_QOS_PROFILES.xml** or a separate XML file).

Figure 4.1: Applications Define Both Their Own and Other Applications' Endpoint Configurations on the next page shows a system with two applications (A and B), both using the LBED plugin. The remote configuration of A is the local configuration of B, since A needs to statically discover B's endpoints and, therefore, it needs to know their information. The same happens in the other direction.

Figure 4.1: Applications Define Both Their Own and Other Applications' Endpoint Configurations




This chapter describes how to configure the LBED Plugin in a generated *Connext* project.

> Only LBED dynamic libraries are available. Therefore, you must link with the dynamic version of the *Connext* libraries. See the RTI Connext Core Libraries Platform Notes for the dynamic libraries associated with your platform.

## 4.1 Generate Connext Project

1. Run **<NDDSHOME>/resource/scripts/rtisetenv_<architecture>.<shell>** in a new command prompt window, to avoid issues with paths and licensing.

   **<architecture>** depends on your target machine (where you will deploy your completed application), and **<shell>** depends on the shell you are using (bash, zsh, bat…). Architecture strings are listed in the RTI Connext Core Libraries Platform Notes. Examples are **x64Win64VS2017** and **x64Linux4gcc7.3.0**.

   (See Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext Getting Started Guide.)
2. Create an IDL file. For example, you may want to use the following (save it as **ExampleLBED.idl**):

```
struct ExampleLBED {
    long data;
};
```

3. Run *RTI Code Generator* (*rtiddsgen*). This guide uses the Modern C++ programming language as reference. The steps are the same for the other supported languages. Feel free to choose the language that best suits your needs. It is recommended you generate an advanced example, if supported by your language and platform; the required code modifications that will be made later in this chapter will be easier if you use the advanced example. For a brief introduction to *Code Generator*, see Run Code Generator, in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext Getting Started Guide. Full details are in the RTI Code Generator User's Manual.

   For Linux® systems:

   ```
   $ rtiddsgen -example <architecture> -language C++11 -exampleTemplate advanced
   ExampleLBED.idl
   ```

   For Windows® systems:

   ```
   > rtiddsgen -example <architecture> -language C++11 -exampleTemplate advanced -
   ppDisable ExampleLBED.idl
   ```

The generated example will be composed of the following files:

**Table 4.1 Modern C++ Files Created for "ExampleLBED.idl"**

| Generated Files | Description |
|---|---|
| ExampleLBED.hpp<br>ExampleLBED.cxx<br>ExampleLBEDPlugin.hpp<br>ExampleLBEDPlugin.cxx | Support for the generated type in C++11. |
| ExampleLBED_publisher.cxx<br>ExampleLBED_subscriber.cxx | Example Publisher and Subscriber applications.<br>Contains a DomainParticipant with a single DataWriter/DataReader for the type defined in ExampleLBED.idl. |
| application.hpp | Provides common utilities to the publisher and subscriber applications, like setting the verbosity and parsing the command-line arguments. |
| README_<architecture>.txt | See this README for instructions on how to open and modify the files. |
| Makefiles and Visual Studio® project files (for Windows applications) | Architecture-dependent build files. |
| USER_QOS_PROFILES.xml | The Quality of Service (QoS) configuration of the DDS entities in the generated example is loaded from this file.<br>The required configuration for using LBED in your application, as well as the static information of the endpoints, can be specified here. |

In the following sections, some of the generated files mentioned above will be modified for using the *Limited Bandwidth Endpoint Discovery Plugin*.

**Note:** The completed XML file that we will be creating in this Getting Started exercise, if you follow the 4.5.1 Using the USER_QOS_PROFILES.xml on page 24 option, can be found in **<path to examples>/connext_dds/c++11/limited_bandwidth_plugins/lbediscovery/USER_QOS_PROFILES.xml**, for your reference.

## 4.2 Add Name to Each DomainParticipant

Each time a *DomainParticipant* is discovered, the LBED Plugin uses the name of that *DomainParticipant* (propagated in the participant announcements) to look for its endpoints' information in the XML file. Since the <participant_name> field is used to associate a participant with its XML information, each *DomainParticipant* using LBED must have a non-NULL name that identifies it.

> **Note:** If you plan on starting multiple applications with participants that have the same configuration and set of endpoints, these participants can reuse the same configuration and have the same name. For example, you could have two temperature sensors with the same *DomainParticipant* and QoS, each with a *DataWriter* on the "Temperature" topic. The only difference would be the data both sensors publish. If you use LBED in this case, the XML of both *DomainParticipants* will be the same; you don't need to duplicate that information and give the *DomainParticipants* different names.

The name of a *DomainParticipant* can be configured using the "ENTITY_NAME QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual. The **USER_QOS_PROFILES.xml** generated in 4.1 Generate Connext Project on page 16 already assigns a name to the *DomainParticipants*. It assigns the same name, **ExampleLBEDParticipant**, to both the *Publisher* and *Subscriber DomainParticipants*; however, to use LBED, *Publisher* and *Subscriber DomainParticipants* need to have different names because they contain different sets of endpoints. Therefore, in this exercise, we will create two separate QoS profiles: one for the *Publisher DomainParticipant* and another for the *Subscriber*, each with a different <participant_name>.

1. Remove the existing qos_profile named "ExampleLBED_Profile" in the **USER_QOS_PROFILES.xml** file and copy and paste the following two profiles:

```
<!-- The QoS profile used by the publishing entities -->
<qos_profile name="ExampleLBED_Publisher_Profile">
    <domain_participant_qos>
        <participant_name>
            <name>ExampleLBEDParticipantPublisher</name>
        </participant_name>
    </domain_participant_qos>
</qos_profile>

<!-- The QoS profile used by the subscribing entities -->
<qos_profile name="ExampleLBED_Subscriber_Profile">
    <domain_participant_qos>
        <participant_name>
            <name>ExampleLBEDParticipantSubscriber</name>
```

```
        </participant_name>
    </domain_participant_qos>
</qos_profile>
```

2. Modify **ExampleLBED_publisher.cxx** so that the *DomainParticipant*, *Topic*, *Publisher*, and *DataWriter* use the "ExampleLBED_Publisher_Profile":

```
dds::domain::DomainParticipant participant(
    domain_id,
    dds::core::QosProvider::Default().participant_qos(
        "ExampleLBED_Library::ExampleLBED_Publisher_Profile"));

...

dds::topic::Topic<ExampleLBED> topic(
    participant,
    "Example ExampleLBED",
    dds::core::QosProvider::Default().topic_qos(
        "ExampleLBED_Library::ExampleLBED_Publisher_Profile"));

...

dds::pub::Publisher publisher(
    participant,
    dds::core::QosProvider::Default().publisher_qos(
        "ExampleLBED_Library::ExampleLBED_Publisher_Profile"));

...

dds::pub::DataWriter<ExampleLBED> writer(
    publisher,
    topic,
    dds::core::QosProvider::Default().datawriter_qos(
        "ExampleLBED_Library::ExampleLBED_Publisher_Profile"),
        listener,
        status_mask);
```

3. Modify **ExampleLBED_subscriber.cxx** so that the *DomainParticipant*, *Topic*, *Subscriber*, and *DataReader* use the "ExampleLBED_Subscriber_Profile":

```
dds::domain::DomainParticipant participant(
    domain_id,
    dds::core::QosProvider::Default().participant_qos(
        "ExampleLBED_Library::ExampleLBED_Subscriber_Profile"));

...

dds::topic::Topic<ExampleLBED> topic(
    participant,
    "Example ExampleLBED",
    dds::core::QosProvider::Default().topic_qos(
        "ExampleLBED_Library::ExampleLBED_Subscriber_Profile"));
```

```
...

dds::sub::Subscriber subscriber(
    participant,
    dds::core::QosProvider::Default().subscriber_qos(
        "ExampleLBED_Library::ExampleLBED_Subscriber_Profile"));

...

dds::sub::DataReader<ExampleLBED> reader(
    subscriber,
    topic,
    dds::core::QosProvider::Default().datareader_qos(
        "ExampleLBED_Library::ExampleLBED_Subscriber_Profile"),
        listener,
        status_mask);
```

## 4.3 Add an RTPS Object ID to Each Endpoint

The RTPS object ID is an integer that uniquely identifies an endpoint of a specific kind (*DataWriter* or *DataReader*) within a *DomainParticipant*. Normally, the RTPS object ID is automatically assigned by *Connext* when an endpoint is created, and it is exchanged during the Simple Endpoint Discovery phase. However, the LBED Plugin replaces Simple Endpoint Discovery, so this auto-assigned RTPS object ID will not be propagated to other endpoints. In order for remote endpoints to be discovered, LBED needs to know the RTPS object ID of a remote participant's entities before discovery initiates.

Therefore, when LBED is used, you must manually define the RTPS object ID of every endpoint in the application. This can be done by means of the "DATA_WRITER_PROTOCOL QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual and "DATA_READER_PROTOCOL QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual. **The RTPS object ID values need to be unique within the same *DomainParticipant* and per endpoint kind (e.g., two *DataWriters* of the same *DomainParticipant* cannot have the same RTPS object ID, but a *DataReader* and a *DataWriter* could have the same value).**

> If your application uses RTI Connext Core Libraries XML-Based Application Creation, you do not have to manually specify the RTPS object ID. In this unique case, LBED is able to automatically infer which RTPS object ID *Connext* will assign to each endpoint. You can see an example of an application that uses LBED and *XML-Based Application Creation* in the rticonnextdds-examples section on GitHub.

1. In your **USER_QOS_PROFILES.xml**, add the following XML snippet to the <qos_profile> named "ExampleLBED_Publisher_Profile" after the <domain_participant_qos> and make sure your *DataWriter* is created using this QoS profile (review the *DataWriter's* creation call in **ExampleLBED_publisher.cxx** and make sure the name of the profile used there is the same as that used here: "ExampleLBED_Publisher_Profile"). 100 is an example value, but you could use

any other value as long as it is unique within the same *DomainParticipant* and per endpoint kind, and falls into the valid range as described for the **rtps_object_id** field in the API Reference HTML documentation.

```
<datawriter_qos>
    <protocol>
        <rtps_object_id>100</rtps_object_id>
    </protocol>
</datawriter_qos>
```

2. In your **USER_QOS_PROFILES.xml**, add the following XML snippet to the <qos_profile> named "ExampleLBED_Subscriber_Profile" after the <domain_participant_qos> and make sure your *DataReader* is created using this QoS profile (review the *DataReader's* creation call in **ExampleLBED_subscriber.cxx** and make sure the name of the profile used there is the same as that used here: "ExampleLBED_Subscriber_Profile"). 200 is an example value, but you could use any other value as long as it is unique within the same *DomainParticipant* (we could have used 100 here as well because the *DataReader* and *DataWriter* are being created by different participants) and per endpoint kind, and falls into the valid range as described for the **rtps_object_id** field in the API Reference HTML documentation.

```
<datareader_qos>
    <protocol>
        <rtps_object_id>200</rtps_object_id>
    </protocol>
</datareader_qos>
```

## 4.4 Enable LBED Plugin on Each DomainParticipant

The next step is to tell each *DomainParticipant* that it should use the *Limited Bandwidth Endpoint Discovery Plugin* instead of Simple Endpoint Discovery. There are two ways of doing this, and both require configuring the *DomainParticipant* QoS in the **USER_QOS_PROFILES.xml** file.

The options described below (4.4.1 Using the Builtin Discovery Plugins Mask (Recommended) below or 4.4.2 Using the LBED Plugin Properties on the next page) are mutually exclusive, which means that it is not possible to enable the plugin for the same *DomainParticipant* using both of these mechanisms at the same time. If you do, an error will be displayed and *DomainParticipant* creation will fail.

### 4.4.1 Using the Builtin Discovery Plugins Mask (Recommended)

The **builtin_discovery_plugins** mask (set in the *DomainParticipant's* "DISCOVERY_CONFIG QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual) can be used to select the built-in discovery plugins a *DomainParticipant* should use. This mask can be used to enable the LBED plugin in a *DomainParticipant* if its value is set to DPSE (**D**ynamic **P**articipant discovery, **S**tatic **E**ndpoint discovery).

Copy and paste the following XML snippet into the <domain_participant_qos> tag of both QoS profiles: "ExampleLBED_Publisher_Profile" and "ExampleLBED_Subscriber_Profile":

```
<discovery_config>
    <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
</discovery_config>
```

Using DPSE automatically configures the *DomainParticipant* properties and QoS in the same way as described in 4.4.2 Using the LBED Plugin Properties below.

Along with the DPSE value, the following optional LBED properties can be specified:

**dds.discovery.endpoint.lbediscovery.config_file**
**dds.discovery.endpoint.lbediscovery.verbosity**
**dds.discovery.endpoint.lbediscovery.property_validation_action**

See Table 4.2 LBED Configuration Properties for Connext for further information.

DPSE (Dynamic Participant discovery, Static Endpoint discovery) and SEDP (Simple Endpoint Discovery Protocol) cannot be specified simultaneously in the **builtin_discovery_plugins** mask.

If you are using both the LBED plugin and the LBPD plugin, this mask needs to be set to MASK_NONE. Therefore, in that case, DPSE cannot be used for enabling LBED. You must use the LBED plugin properties to enable it (see 4.4.2 Using the LBED Plugin Properties below).

## 4.4.2 Using the LBED Plugin Properties

*Connext* provides a mechanism to dynamically load an external plugin from an XML QoS profile, i.e., the **USER_QOS_PROFILES.xml** file we have been modifying. That mechanism is the "PROPERTY QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual.

The PROPERTY QoS policy stores name/value (string) pairs that can be used to configure certain parameters of *Connext* that are not exposed through formal QoS policies. *Connext* uses this mechanism, for example, to configure external transports and plugins.

To enable LBED using these properties, add the following XML snippet into the <domain_participant_qos> tag of both QoS profiles “ExampleLBED_Publisher_Profile" and “ExampleLBED_Subscriber_Profile”. This snippet shows the minimum required properties for enabling the plugin:

```
<property>
    <value>
        <element>
            <name>dds.discovery.endpoint.lbediscovery.library</name>
            <value>rtilbedisc</value>
        </element>
        <element>
            <name>dds.discovery.endpoint.lbediscovery.create_function</name>
            <value>DDS_LBEDiscoveryPlugin_create</value>
```

```
        </element>
        <element>
            <name>dds.discovery.endpoint.load_plugins</name>
            <value>dds.discovery.endpoint.lbediscovery</value>
        </element>
    </value>
</property>
```

For more information about the meaning of the properties above, their values, and other LBED properties that can be specified, see Table 4.2 LBED Configuration Properties for Connext.

In addition to setting the properties described above, set the **builtin_discovery_plugins mask** (using the *DomainParticipant's* "DISCOVERY_CONFIG QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual) to SPDP as shown in the XML excerpt below. The default value of this mask is SDP (Simple Discovery Protocol), which consists of two parts, Simple Participant Discovery Protocol (SPDP) and Simple Endpoint Discovery Protocol (SEDP). Set the **builtin_discovery_plugins** to just SPDP to indicate that SPDP is the only default plugin you'll be using, since you will be replacing the SEDP portion with the LBED plugin:

```
<discovery_config>
    <builtin_discovery_plugins>SPDP</builtin_discovery_plugins>
</discovery_config>
```

**If you are using both the LBED plugin and the LBPD plugin, set this mask to MASK_NONE. An example of this configuration is shown in <path to examples>/connext_dds/c++11/limited_bandwidth_plugins/dil-stacking.**

## 4.5 Add the DDS-XML Definition of the Entities

If you followed the previous steps in this chapter, you should have an application in which each *DomainParticipant* has a non-null name, each created endpoint has a predefined RTPS object ID, and both *DomainParticipants* use the LBED Plugin for the Endpoint Discovery phase. However, if you were to run the *Publisher* and the *Subscriber* now, you would see error messages similar to the following:

```
$ ./ExampleLBED_subscriber
…
ERROR [0x01016E46,0x8484F58C,0x5EBD43BE:0x000100C7{Entity=DR,MessageKind=DATA}|RECEIVE FROM
0x00000000,0x00000000,0x00000000:0x000100C2|REGISTER DP
"ExampleLBEDParticipantPublisher"|LC:DISC]DDS_LBEDiscoveryPlugin_registerParticipant:The
discovered DomainParticipant [guid=0x01013049,0xFA1254B0,0x6D982A45:0x000001C1] is not in
the file.
ERROR [0x01016E46,0x8484F58C,0x5EBD43BE:0x000100C7{Entity=DR,MessageKind=DATA}|RECEIVE FROM
0x00000000,0x00000000,0x00000000:0x000100C2|LC:DISC]DDS_LBEDiscoveryPlugin_
afterRemoteParticipantEnabled:The DomainParticipant "ExampleLBEDParticipantPublisher" cannot
be registered.
…
```

```
$ ./ExampleLBED_publisher
…
ERROR [0x01013049,0xFA1254B0,0x6D982A45:0x000100C7{Entity=DR,MessageKind=DATA}|RECEIVE FROM
```

```
0x00000000,0x00000000,0x00000000:0x000100C2|REGISTER DP
"ExampleLBEDParticipantSubscriber"|LC:DISC]DDS_LBEDiscoveryPlugin_registerParticipant:The
discovered DomainParticipant [guid=0x01016E46,0x8484F58C,0x5EBD43BE:0x000001C1] is not in
the file.
ERROR [0x01013049,0xFA1254B0,0x6D982A45:0x000100C7{Entity=DR,MessageKind=DATA}|RECEIVE FROM
0x00000000,0x00000000,0x00000000:0x000100C2|LC:DISC]DDS_LBEDiscoveryPlugin_
afterRemoteParticipantEnabled:The DomainParticipant "ExampleLBEDParticipantSubscriber"
cannot be registered.
…
```

In addition, no data is exchanged between the *Publisher* and the *Subscriber* applications. This is because the endpoints are not being discovered. We are missing the most important requirement of the *Limited Bandwidth Endpoint Discovery Plugin*: the XML definition of the endpoints that should be discovered (the remote configuration).

The *Publisher* requires the XML definition of the *Subscriber's* endpoints. When the *Publisher* discovers the *Subscriber DomainParticipant*, the *Publisher* will use the name of the *Subscriber* to look for its definition in the XML file, and will register all the *Subscriber's* endpoints with their associated QoS, *Topics*, and types. In the same way, the *Subscriber* requires the XML definition of the *Publisher's* endpoints. The error messages above tell us that the definition of "ExampleLBEDParticipantSubscriber" and "ExampleLBEDParticipantPublisher" could not be found in the XML file (by default, the **USER_QOS_PROFILES.xml** if the **config_file** property in Table 4.2 LBED Configuration Properties for Connext was not specified). Therefore, their endpoints could not be registered.

To define the *Publisher* and *Subscriber DomainParticipants* information, the LBED Plugin uses the OMG's DDS Consolidated XML Syntax (DDS-XML), a standard that determines how to represent and describe DDS entities and resources using XML. This is the same standard that *XML-Based Application Creation* uses. For more information about DDS-XML, it is recommended to read "XML Tags for Configuring Entities" in the RTI Connext Core Libraries XML-Based Application Creation Getting Started Guide or the OMG's DDS Consolidated XML Syntax (DDS-XML) specification.

There are two ways of specifying the DDS-XML definition of the entities for LBED: directly in the **USER_QOS_PROFILES.xml** (in 4.5.1 Using the USER_QOS_PROFILES.xml below) or by creating a separate XML file for each *DomainParticipant* (in 4.5.2 Using a Separate XML File on page 31). **Choose one way for your application.**

## 4.5.1 Using the USER_QOS_PROFILES.xml

If the **config_file** property was not specified when enabling LBED in a *DomainParticipant* (see 4.4.2 Using the LBED Plugin Properties on page 22), the plugin will look for the DDS-XML definition of the endpoints in **USER_QOS_PROFILES.xml** by default. By using this approach, no additional XML files are required. Your **USER_QOS_PROFILES.xml** will contain the definition of the QoS used by your entities, as well as the DDS-XML static definition of the endpoints required by LBED. In other words, the file will contain both local and remote configurations.

LBED needs to know the following information about the entities:

- For each *DomainParticipant* using LBED whose endpoints need to be statically-discovered, LBED must know the *DomainParticipant's* topology: how many *Publishers/Subscribers* it has, how many *DataWriters/DataReaders* there are per *Publisher/Subscriber*, and if the endpoints are using an implicit *Publisher/Subscriber*.
- For the entities above, which QoS they are using.
- The *Topic* to which each endpoint is attached.
- The type that each *Topic* uses.

You will learn how to add each one of these requirements to the **USER_QOS_PROFILES.xml** file in the following sections.

**To get XML file validation and autocompletion during editing, change the USER_QOS_PROFILE.xml's XSD from rti_dds_qos_profiles.xsd to rti_dds_profiles.xsd:**

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///<NDDSHOME>/resource/schema/rti_dds_profiles.xsd">
...
```

#### 4.5.1.1 Add the types used by the endpoints

Your application *Topics* use the ExampleLBED type that was generated from the IDL in 4.1 Generate Connext Project on page 16. You need to specify the XML type definition of that type in the **USER_QOS_PROFILES.xml** file using the <types> tag.

> If your application uses a Built-in Data Type instead (see "Built-in Data Types" in the RTI Connext Core Libraries User's Manual), this definition is not required and you can skip this step.

It is possible to convert from an IDL file to XML using *Code Generator* (*rtiddsgen*):

```
$ rtiddsgen -convertToXml ExampleLBED.idl
```

This will generate a file called **ExampleLBED.xml**. Copy and paste the <types> tag and its content into the **USER_QOS_PROFILES.xml** after the <qos_library> section:

```
<types>
    <struct name= "ExampleLBED">
        <member name="data" type="int32"/>
    </struct>
</types>
```

> **This step is optional but recommended when using user-defined types.**
>
> If the definition of the type is not provided in the XML file, the LBED plugin will automatically check if there is a type with the same name registered in the *DomainParticipant* of the local application (the one that created the plugin instance) and will use its definition. This check

works out if the type in the local *DomainParticipant* is the same as the one used by the endpoints that need to be discovered, which would be true in correct system configurations. But if you have different types with the same name, you may get undefined behaviors in LBED (e.g., no discovery).

Therefore, if you decide to not add the XML type definition, to avoid errors, it is recommended that you create the *DomainParticipant* disabled, register the type in the application, and then enable the *DomainParticipant*. That way, you ensure LBED does not start registering endpoints until the type has been registered in the participant.

To create a *DomainParticipant* disabled, set **autoenable_created_entities** in the DomainParticipantFactory ENTITYFACTORY QosPolicy to false. See "Enabling DDS Entities" in the RTI Connext Core Libraries User's Manual for information about how to enable the *DomainParticipant*. See "Data Types and DDS Data Samples" in the RTI Connext Core Libraries User's Manual for information about registering types.

#### 4.5.1.2 Add the Topics

The endpoints of the generated application use a single *Topic* called "Example ExampleLBED". In DDS-XML, *Topics* are specified inside a domain tag which, in turn, is specified inside a domain_library. This exercise assumes the application uses domain ID 0, but you can use whichever ID you prefer.

Copy and paste the following XML snippet in your **USER_QOS_PROFILES.xml** file after the types tag you added in the previous step (modify the domain_id attribute if you want to use another ID):

```
<domain_library name="ExampleLBED_Domain_Library">
    <domain name="ExampleLBED_Domain" domain_id="0">
        <register_type name="ExampleLBED" type_ref="ExampleLBED"/>
        <topic name="Example ExampleLBED" register_type_ref="ExampleLBED"/>
    </domain>
</domain_library>
```

Regarding the snippet above:

- The names of the domain_library and domain tags are just examples. You can use any names you like.
- Make sure the name attribute of the topic tag is the same as the name with which the *Topic* is created in the application.
- The LBED plugin uses the register_type tag for associating a *Topic* with its XML type definition. Therefore:
    - If your application uses a Built-in Data Type instead (see "Built-in Data Types" in the RTI Connext Core Libraries User's Manual), the register_type tag is not required and you can put the name of the Built-in Data Type directly in the *Topic's* register_type_ref attribute.

For example:

```
<domain name="ExampleLBED_Domain" domain_id="0">
    <topic name="Example ExampleLBED" register_type_ref="DDS::String"/>
</domain>
```

- **If you decided to not add the XML type definition to the file** in 4.5.1.1 Add the types used by the endpoints on page 25, the register_type tag is not required and you can put the name of your type directly in the *Topic's* register_type_ref attribute. For example:

```
<domain name="ExampleLBED_Domain" domain_id="0">
    <topic name="Example ExampleLBED" register_type_ref="ExampleLBED"/>
</domain>
```

- The register_type tag is also used for specifying the name with which the type referenced by type_ref (which must be an existing type under the types tag) is registered in the application. In this case, the XML type definition you added in 4.5.1.1 Add the types used by the endpoints on page 25 (named "ExampleLBED") is registered in the application with the same name. You must make sure the name attribute of the register_type tag is the same as the name with which the type is registered in the application code.

### 4.5.1.3 Add the DomainParticipants' topology

The generated application has two *DomainParticipants*: the ExampleLBEDParticipantPublisher and ExampleLBEDParticipantSubscriber. If you open the **ExampleLBED_publisher.cxx** program, you will see that ExampleLBEDParticipantPublisher has a single *Publisher* and a single *DataWriter*. The same happens for ExampleLBEDParticipantSubscriber: it has a *Subscriber* and a *DataReader*. Both end-points use the "Example ExampleLBED" *Topic* and the ExampleLBED type.

The information above is what we call "topology," and we need to represent it using DDS-XML. For that, use the domain_participant_library. Copy and paste the following XML snippet in your **USER_QOS_PROFILES.xml** file after the domain_library tag you added in the previous step:

```
<domain_participant_library name="ExampleLBED_DomainParticipant_Library">
    <domain_participant name="ExampleLBEDParticipantPublisher"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <publisher name="Pub">
            <data_writer name="DW" topic_ref="Example ExampleLBED"></data_writer>
        </publisher>
    </domain_participant>

    <domain_participant name="ExampleLBEDParticipantSubscriber"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <subscriber name="Sub">
            <data_reader name="DR" topic_ref="Example ExampleLBED"></data_reader>
        </subscriber>
    </domain_participant>
</domain_participant_library>
```

Regarding the snippet above:

- The name of the domain_participant_library is just an example. You can use any name you like.
- Make sure the name attributes of the domain_participant tags are the same as the names you specified for each *DomainParticipant* in 4.2 Add Name to Each DomainParticipant on page 18. This is the field that LBED uses to look for the *DomainParticipants'* definition in the XML file.
- Make sure the domain_ref attribute of each *DomainParticipant* links to the correct domain tag added in 4.5.1.2 Add the Topics on page 26 (that is, the domain that these participants use).
- Although LBED does not use the name attribute for *Publishers*, *Subscribers*, and endpoints, it is required by the DDS-XML standard, and *Connext*'s internal XML parser imposes a restriction on the name attribute:

  The name attribute must be unique within the same parent tag (e.g., two <data_writer> elements under the same <publisher> cannot have the same name attribute).

  **Note:** If your application has two or more entities that are essentially the same (e.g., your *Publisher* has two or more identical *DataWriters* with the same QoS, *Topic*, and type) and you are using *XML-Based Application Creation*, then you don't need to duplicate the information of the same entity several times in DDS-XML but with different names. You can use the multiplicity attribute instead to indicate how many entities use the same configuration (see the "Domain Participant Tag" table, in the "Participant Library" section, of the RTI Connext Core Libraries XML-Based Application Creation Getting Started Guide for more information about the multiplicity attribute, which applies to *Publishers*, *Subscribers*, and endpoints). If you are using LBED without *XML-Based Application Creation*, then you need to list each endpoint separately (with a different name attribute) and assign a different RTPS object ID to each one.

- Make sure the topic_ref attribute of each endpoint links to the correct *Topic* tag added in 4.5.1.2 Add the Topics on page 26 (that is, the *Topic* these endpoints use in the application).
- If your endpoints use the implicit *Publisher* or *Subscriber*, then the publisher or subscriber tag is not required. The data_reader and data_writer tags can live directly under the domain_participant tag.

#### 4.5.1.4 Add the required QoS profiles to the entities

In the DDS-XML definition from the previous step, no entity specifies which QoS profile it is using, so the default QoS profile (a profile marked as is_default_qos="true" or the default QoS values if there is none) will be assumed for all of them. However, if any of your entities use a QoS profile different than the default one and it contains information that LBED needs to know (see 4.10 Supported QoS on page 43 for more information about which QoS policies LBED needs to be aware of), you must explicitly define the QoS profile that entity is using in the XML file.

For example, in this exercise, only the endpoints use a non-default QoS profile that specifies something LBED needs to know (the **rtps_object_id**), so we need to specify in DDS-XML which profiles the endpoints are using. You must make sure that the QoS profile you specify for a given entity is the same

one that this entity uses when it is created in the application, because QoS policies are used for determining the matching between entities.

This is the same XML snippet from the previous section. See the QoS profiles in bold that you need to add:

```
<domain_participant_library name="ExampleLBED_DomainParticipant_Library">
    <domain_participant name="ExampleLBEDParticipantPublisher"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <publisher name="Pub">
            <data_writer name="DW" topic_ref="Example ExampleLBED">
                <datawriter_qos
                    base_name="ExampleLBED_Library::ExampleLBED_Publisher_Profile"/>
            </data_writer>
        </publisher>
    </domain_participant>

<domain_participant name="ExampleLBEDParticipantSubscriber"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <subscriber name="Sub">
            <data_reader name="DR" topic_ref="Example ExampleLBED">
                <datareader_qos
                    base_name="ExampleLBED_Library::ExampleLBED_Subscriber_Profile"/>
            </data_reader>
        </subscriber>
    </domain_participant>
</domain_participant_library>
```

After following this tutorial, your XML file should look similar to the following. This file can also be found in **<path to examples>/connext_dds/c++11/limited_bandwidth_plugins/lbediscovery/USER_QOS_PROFILES.xml**.

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///<NDDSHOME>/resource/schema/rti_dds_profiles.xsd">

    <qos_library name="ExampleLBED_Library">

        <!-- The QoS profile used by the publishing entities -->
        <qos_profile name="ExampleLBED_Publisher_Profile">
            <domain_participant_qos>
                <discovery_config>
                    <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
                </discovery_config>
                <participant_name>
                    <name>ExampleLBEDParticipantPublisher</name>
                </participant_name>
            </domain_participant_qos>
            <datawriter_qos>
                <protocol>
                    <rtps_object_id>100</rtps_object_id>
                </protocol>
            </datawriter_qos>
```

```
    </qos_profile>

    <!-- The QoS profile used by the subscribing entities -->
    <qos_profile name="ExampleLBED_Subscriber_Profile">
        <domain_participant_qos>
            <discovery_config>
                <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
            </discovery_config>
            <participant_name>
                <name>ExampleLBEDParticipantSubscriber</name>
            </participant_name>
        </domain_participant_qos>
        <datareader_qos>
            <protocol>
                <rtps_object_id>200</rtps_object_id>
            </protocol>
        </datareader_qos>
    </qos_profile>

</qos_library>

<types>
    <struct name= "ExampleLBED">
        <member name="data" type="int32"/>
    </struct>
</types>

<domain_library name="ExampleLBED_Domain_Library">
<!-- IMPORTANT: change the domain_id value if you plan to use a
     domain different than 0 -->
    <domain name="ExampleLBED_Domain" domain_id="0">
        <register_type name="ExampleLBED" type_ref="ExampleLBED"/>
        <topic name="Example ExampleLBED" register_type_ref="ExampleLBED"/>
    </domain>
</domain_library>

<domain_participant_library name="ExampleLBED_DomainParticipant_Library">
    <domain_participant name="ExampleLBEDParticipantPublisher"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <publisher name="Pub">
            <data_writer name="DW" topic_ref="Example ExampleLBED">
                <datawriter_qos
                    base_name="ExampleLBED_Library::ExampleLBED_Publisher_Profile"/>
            </data_writer>
        </publisher>
    </domain_participant>

    <domain_participant name="ExampleLBEDParticipantSubscriber"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <subscriber name="Sub">
            <data_reader name="DR" topic_ref="Example ExampleLBED">
                <datareader_qos
                    base_name="ExampleLBED_Library::ExampleLBED_Subscriber_Profile"/>
```

```
                </data_reader>
            </subscriber>
        </domain_participant>
    </domain_participant_library>
</dds>
```

## 4.5.2 Using a Separate XML File

The **config_file** property (see Table 4.2 LBED Configuration Properties for Connext) can be used to specify a separate XML file (different from **USER_QOS_PROFILES.xml**) that exclusively contains the DDS-XML static definition of the endpoints that LBED requires. By following this approach, your **USER_QOS_PROFILES.xml** file will contain the local configuration and the separate XML file the remote one.

The structure and information of that XML file is the same as explained in 4.5.1 Using the USER_QOS_PROFILES.xml on page 24. It is highly recommended to read that section before continuing with the steps below. The only difference is that you don't reuse the **USER_QOS_PROFILES.xml** file for LBED and you use a dedicated XML file instead.

In this exercise, we have chosen to create a separate XML file for each *DomainParticipant* (one for the *Publisher* and another for the *Subscriber*) but, if you prefer, you can create a single file that contains the LBED-required information for both.

The *Publisher DomainParticipant* will load the XML file with the information of the *Subscriber DomainParticipant* endpoints, and the *Subscriber DomainParticipant* will load the XML file with the information of the *Publisher*:

1. Create two new XML files in the working directory. For example, you may want to name them **LBEDPublisher.xml** and **LBEDSubscriber.xml**. The first one will contain the information of the *Publisher* participant endpoints and, the second one, the information of the *Subscriber*.
2. Modify the "ExampleLBED_Publisher_Profile" in **USER_QOS_PROFILES.xml**, adding the **config_file** property to the *DomainParticipant* QoS with the path to the **LBEDSubscriber.xml** file as value. Add it anywhere inside the <domain_participant_qos> tag.

```
<property>
    <value>
        <element>
            <name>dds.discovery.endpoint.lbediscovery.config_file</name>
            <value>LBEDSubscriber.xml</value>
        </element>
    </value>
</property>
```

3. Modify the "ExampleLBED_Subscriber_Profile" in **USER_QOS_PROFILES.xml**, adding the **config_file** property to the *DomainParticipant* QoS with the path to the **LBEDPublisher.xml** file as value. Add it anywhere inside the <domain_participant_qos> tag.

```
<property>
    <value>
        <element>
            <name>dds.discovery.endpoint.lbediscovery.config_file</name>
            <value>LBEDPublisher.xml</value>
        </element>
    </value>
</property>
```

4. Start by creating the <dds> tag in both XML files (replace <NDDSHOME> with the actual path to your *Connext* installation directory):

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///<NDDSHOME>/resource/schema/rti_dds_
profiles.xsd">

</dds>
```

5. In **LBEDPublisher.xml**, create a qos_library and copy-paste "ExampleLBED_Publisher_Profile" from **USER_QOS_PROFILES.xml**, since this is the profile that the entities of the *Publisher* participant use. Rename it, for example, to “ExampleLBED_Publisher_Profile_Static”. This is because your application will load both the **USER_QOS_PROFILES.xml** and **LBEDPublisher.xml** files, and there cannot be two profiles with the same name.

   You can remove the <domain_participant_qos> element altogether, since the information it contains doesn’t need to be known by LBED (see 4.10 Supported QoS on page 43 for more information about which QoS policies LBED needs to know). Your profile should look similar to the following:

```
<qos_library name="ExampleLBED_Library">
    <qos_profile name="ExampleLBED_Publisher_Profile_Static">
        <datawriter_qos>
            <protocol>
                <rtps_object_id>100</rtps_object_id>
            </protocol>
        </datawriter_qos>
    </qos_profile>
</qos_library>
```

6. Do the same for **LBEDSubscriber.xml**: create a qos_library and copy-paste the "ExampleLBED_Subscriber_Profile" from **USER_QOS_PROFILES.xml**. Rename it to "ExampleLBED_Subscriber_Profile_Static" and remove the non-required information:

```
<qos_library name="ExampleLBED_Library">
    <qos_profile name="ExampleLBED_Subscriber_Profile_Static">
        <datareader_qos>
            <protocol>
                <rtps_object_id>200</rtps_object_id>
            </protocol>
        </datareader_qos>
    </qos_profile>
```

```
</qos_library>
```

7. Since both *Topics* use the same type (ExampleLBED), copy and paste the ExampleLBED XML type definition (see 4.5.1.1 Add the types used by the endpoints on page 25) to both XML files.
8. Since both endpoints use the same *Topic* (Example ExampleLBED), copy and paste the DDS-XML snippet from 4.5.1.2 Add the Topics on page 26 (the contents of the <domain_library> tag) to both XML files. Don't forget to modify the domain_id attribute if you plan to use another ID.
9. Add a <domain_participant_library> and the *Publisher DomainParticipant's* topology to **LBEDPublisher.xml** (see the XML snippet in 4.5.1.3 Add the DomainParticipants' topology on page 27):

```
<domain_participant_library name="ExampleLBED_DomainParticipant_Library">
    <domain_participant name="ExampleLBEDParticipantPublisher"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <publisher name="Pub">
            <data_writer name="DW" topic_ref="Example ExampleLBED"></data_writer>
        </publisher>
    </domain_participant>
</domain_participant_library>
```

10. Add a <domain_participant_library> and the *Subscriber DomainParticipant's* topology to **LBEDSubscriber.xml** (see the XML snippet in 4.5.1.3 Add the DomainParticipants' topology on page 27):

```
<domain_participant_library name="ExampleLBED_DomainParticipant_Library">
    <domain_participant name="ExampleLBEDParticipantSubscriber"
            domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
        <subscriber name="Sub">
            <data_reader name="DR" topic_ref="Example ExampleLBED"></data_reader>
        </subscriber>
    </domain_participant>
</domain_participant_library>
```

11. Add the QoS profiles that the entities use, as explained in 4.5.1.4 Add the required QoS profiles to the entities on page 28, to both XML files. Keep in mind that you renamed the profiles (in step 5, above), so the specified base_name must match the new names.

This is the final **LBEDPublisher.xml** file (replace <NDDSHOME> with the actual path to your *Connext* installation directory):

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="file:///<NDDSHOME>/resource/schema/rti_dds_
profiles.xsd">
    <qos_library name="ExampleLBED_Library">
        <qos_profile name="ExampleLBED_Publisher_Profile_Static">
            <datawriter_qos>
                <protocol>
                    <rtps_object_id>100</rtps_object_id>
                </protocol>
```

```
            </datawriter_qos>
        </qos_profile>
    </qos_library>

    <types>
        <struct name="ExampleLBED">
            <member name="data" type="int32" />
        </struct>
    </types>

    <domain_library name="ExampleLBED_Domain_Library">
        <domain name="ExampleLBED_Domain" domain_id="0">
            <register_type name="ExampleLBED" type_ref="ExampleLBED" />
            <topic name="Example ExampleLBED" register_type_ref="ExampleLBED" />
        </domain>
    </domain_library>

    <domain_participant_library name="ExampleLBED_DomainParticipant_Library">
        <domain_participant name="ExampleLBEDParticipantPublisher"
                domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
            <publisher name="Pub">
                <data_writer name="DW" topic_ref="Example ExampleLBED">
                    <datawriter_qos
                        base_name="ExampleLBED_Library::ExampleLBED_Publisher_Profile_
Static" />
                </data_writer>
            </publisher>
        </domain_participant>
    </domain_participant_library>
</dds>
```

This is the final **LBEDSubscriber.xml** file (replace <NDDSHOME> with the actual path to your *Connext* installation directory):

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="file:///<NDDSHOME>/resource/schema/rti_dds_
profiles.xsd">
    <qos_library name="ExampleLBED_Library">
        <qos_profile name="ExampleLBED_Subscriber_Profile_Static">
            <datareader_qos>
                <protocol>
                    <rtps_object_id>200</rtps_object_id>
                </protocol>
            </datareader_qos>
        </qos_profile>
    </qos_library>

    <types>
        <struct name="ExampleLBED">
            <member name="data" type="int32" />
        </struct>
    </types>

    <domain_library name="ExampleLBED_Domain_Library">
```

```
        <domain name="ExampleLBED_Domain" domain_id="0">
            <register_type name="ExampleLBED" type_ref="ExampleLBED" />
            <topic name="Example ExampleLBED" register_type_ref="ExampleLBED" />
        </domain>
    </domain_library>

    <domain_participant_library name="ExampleLBED_DomainParticipant_Library">
        <domain_participant name="ExampleLBEDParticipantSubscriber"
                domain_ref="ExampleLBED_Domain_Library::ExampleLBED_Domain">
            <subscriber name="Sub">
                <data_reader name="DR" topic_ref="Example ExampleLBED">
                    <datareader_qos
                        base_name="ExampleLBED_Library::ExampleLBED_Subscriber_Profile_
Static" />
                </data_reader>
            </subscriber>
        </domain_participant>
    </domain_participant_library>
</dds>
```

# 4.6 Run the Applications

The last step is checking that your applications' endpoints discover each other statically and that there is communication. Open two command prompts and configure the environment in both to point to the location of the dynamic libraries, as explained below.

## 4.6.1 Configure the Environment in Both Command Prompts

Configure the environment for *Connext* by running **<NDDSHOME>/resource/scripts/rtisetenv_<architecture>.<shell>** in both command prompts, one for the *Publisher* and one for the *Subscriber*. See Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext Getting Started Guide. This will update your PATH to include the location of the *Connext* binaries.

## 4.6.2 Check Communication

1. Run the *Publisher* in one of the command prompts and the *Subscriber* in the other.

   If you configured a domain ID different than 0, pass the **-d** <your domain ID> argument to both binaries.
2. You should see the received data in the *Subscriber's* command prompt, which indicates successful communication:

For Linux systems:

*Publisher*:

```
$ ./objs/<architecture>/ExampleLBED_publisher
Writing ExampleLBED, count 0
Writing ExampleLBED, count 1
Writing ExampleLBED, count 2
Writing ExampleLBED, count 3
Writing ExampleLBED, count 4
Writing ExampleLBED, count 5
…
```

*Subscriber*:

```
$ ./objs/<architecture>/ExampleLBED_subscriber
ExampleLBED subscriber sleeping up to 1 sec...
[data: 0]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 1]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 2]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 3]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 4]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 5]
ExampleLBED subscriber sleeping up to 1 sec...
…
```

For Windows systems:

*Publisher*:

```
> objs\<architecture>\ExampleLBED_publisher.exe
Writing ExampleLBED, count 0
Writing ExampleLBED, count 1
Writing ExampleLBED, count 2
Writing ExampleLBED, count 3
Writing ExampleLBED, count 4
Writing ExampleLBED, count 5
…
```

*Subscriber*:

```
> objs\<architecture>\ExampleLBED_subscriber.exe
ExampleLBED subscriber sleeping up to 1 sec...
[data: 0]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 1]
ExampleLBED subscriber sleeping up to 1 sec...
```

```
ExampleLBED subscriber sleeping up to 1 sec...
[data: 2]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 3]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 4]
ExampleLBED subscriber sleeping up to 1 sec...
ExampleLBED subscriber sleeping up to 1 sec...
[data: 5]
ExampleLBED subscriber sleeping up to 1 sec...
…
```

## 4.6.3 Check that Your Endpoints Discover Each Other Statically

When LBED is used, endpoints do not exchange publication/subscription DATAs. The information that they normally provide about the endpoints that should be discovered is gathered instead from the XML file. Therefore, to verify that your applications' endpoints are discovering each other using the LBED plugin, and without sending publication/subscription DATAs, we will modify your applications to use only the UDPv4 transport so we can inspect the exchanged packages using Wireshark:

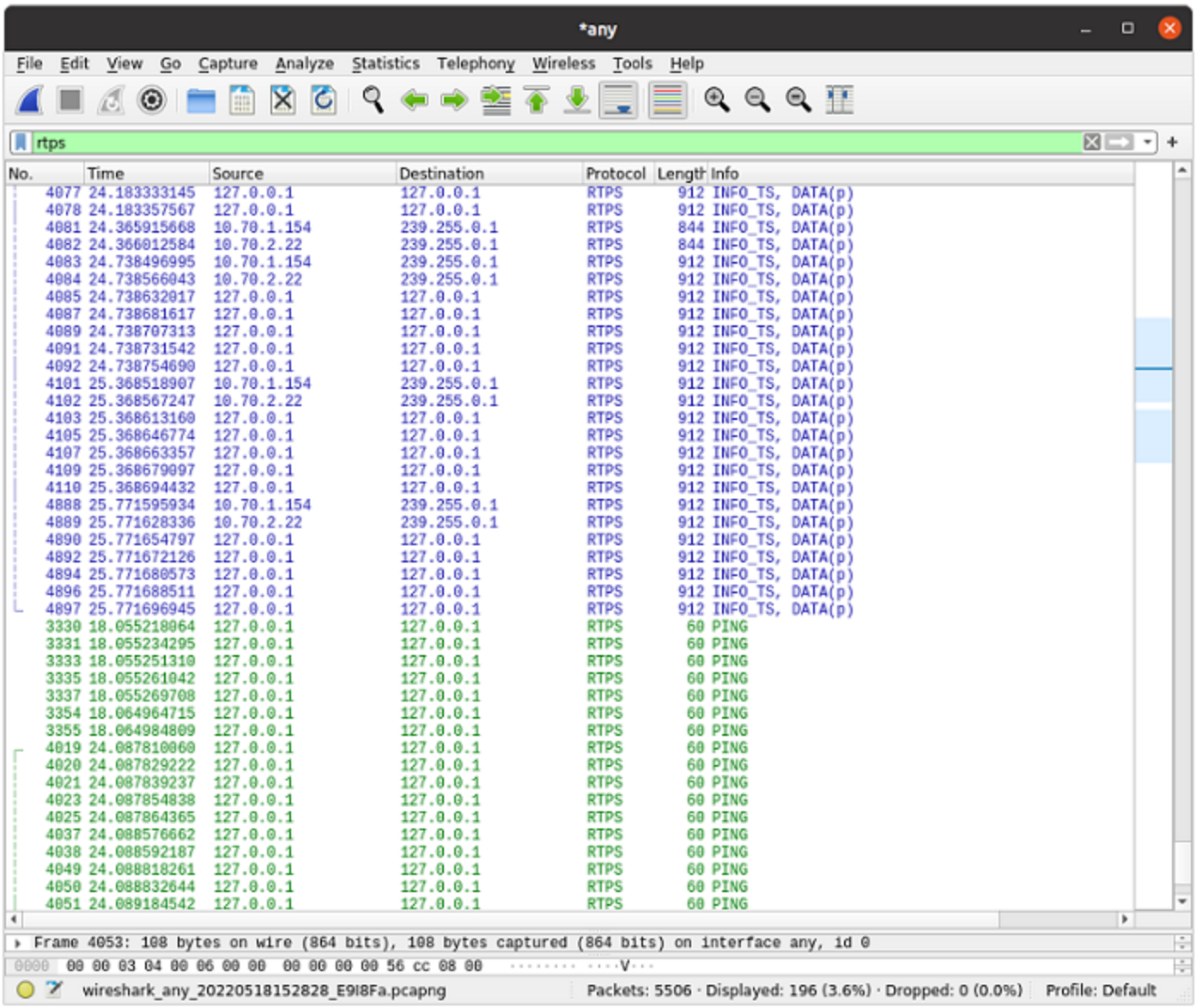
1. In your **USER_QOS_PROFILES.xml** file, add the following XML snippet to both "ExampleLBED_Publisher_Profile" and "ExampleLBED_Subscriber_Profile" in the <domain_participant_qos> tag. You can remove it once you verify LBED is working. For example, in the "ExampleLBED_Subscriber_Profile":

   ```
   <qos_profile name="ExampleLBED_Subscriber_Profile">
       <domain_participant_qos>
           <transport_builtin>
               <mask>UDPv4</mask>
           </transport_builtin>
           ...
       </domain_participant_qos>
       ...
   </qos_profile>
   ```

2. Run Wireshark and start capturing all of your interfaces. Filter captured packages by "rtps". See Using Wireshark with RTI Connext Systems for more information.
3. Run your *Publisher* and *Subscriber* applications as explained in previous sections.

Publication DATAs are shown in Wireshark as DATA(w) and subscription DATAs as DATA(r). If LBED is correctly configured, you should not see any of them in your Wireshark capture:

Figure 4.2: Wireshark Capture of Discovery Traffic when LBED is Being Used




# 4.7 Troubleshooting

## 4.7.1 System does not Recognize or Find rtiddsgen Command

Make sure *Connext* is installed correctly.

Make sure you've correctly set the PATH environment variable (your PATH should include **<NDDSHOME>/bin**). You can run **<NDDSHOME>/resource/scripts/rtisetenv_<architecture>.<shell>** to add the location of the *Connext* binaries to your PATH (see Set Up Environment

Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext Getting Started Guide).

### 4.7.2 rtiddsgen Displays Error (Usually in Windows systems)

If you get this error:

```
The preprocessor 'CL.EXE' cannot be found in your path.
```

Make sure your toolchain's preprocessor is available. If you are using Visual Studio®, make sure you are using a Visual Studio command prompt. Alternatively, run *rtiddsgen* with the **-ppDisable** option.

### 4.7.3 rtiddsgen Gives Warnings

```
File exists and will not be overwritten
```

Some files that would normally be generated already exist and will not be overwritten. These errors are expected.

### 4.7.4 Running Publisher/Subscriber Produces Errors

If you get this error:

```
error while loading shared libraries: libnddscpp2.so: cannot open shared object file: No
such file or directory
```

Make sure the shared library environment variable includes the directory where the *Connext* libraries reside. See 4.6.1 Configure the Environment in Both Command Prompts on page 35.

If you get this error:

```
!open library=librtilbedisc.so
```

Make sure the Limited Bandwidth Plugins bundles provided by RTI are installed correctly. In addition, make sure the shared library environment variable includes the directory where the *Connext* libraries reside. See 4.6.1 Configure the Environment in Both Command Prompts on page 35.

If you get these errors:

```
The discovered DomainParticipant [guid=...] is not in the file.
```

```
The DomainParticipant "..." cannot be registered.
```

LBED is not able to find the DDS-XML definition of a discovered *DomainParticipant*. Make sure the DDS-XML information of that participant is present in the XML file and that the "name" attribute of the <domain_participant> tag matches the one specified in the "ENTITY_NAME QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual.

You may be encountering the following issue: 4.11.1 LBED not able to automatically load USER_QOS_PROFILES.xml if utilized API uses QosProvider and the file is not placed in default locations on page 46.

This may not necessarily be an error: your application could be discovering an interfering *DomainParticipant* in your network that is running in the same domain.

### 4.7.5 Communication does not Occur between Publisher/Subscriber ([data: …] Messages not Displayed in Subscriber Prompt)

- Make sure the *Publisher* participant and the *DataWriter* are created using the "ExampleLBED_Publisher_Profile" (review the entities' creation call in **ExampleLBED_publisher.cxx** and make sure the names of the used profiles are the correct ones).
- Make sure the *Subscriber* participant and the *DataReader* are created using the "ExampleLBED_Subscriber_Profile" (review the entities' creation call in **ExampleLBED_subscriber.cxx** and make sure the names of the used profiles are the correct ones).
- Make sure the specified DDS-XML information is consistent with your applications (e.g., the *Topic*, types, and QoS specified for each endpoint in the XML are the ones that the endpoints are actually using in the applications' code).
- Make sure that you are not missing any of the steps mentioned in this exercise.
- Make sure the names specified through the "ENTITY_NAME QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual for each *DomainParticipant* match the ones specified in DDS-XML. If LBED is not able to find the DDS-XML definition of a discovered *DomainParticipant*, you will see error messages similar to the ones shown in 4.5 Add the DDS-XML Definition of the Entities on page 23.
- If you are using separate XML files for LBED, make sure the *Publisher* is loading the information of the *Subscriber* and vice versa.
- If you modified the QoS of the endpoints, make sure they are compatible. See "QoS Requested vs. Offered Compatibility" in the RTI Connext Core Libraries User's Manual. You can debug if the QoS are incompatible by setting the *Connext* verbosity (not the LBED one) to WARNING. In the generated Modern C++ applications, use the **-v 2** argument to set a WARNING verbosity.

## 4.8 Limitations

LBED relies exclusively on the information that is present in the XML file. Therefore, it is not able to infer anything from the endpoints that requires information that cannot be specified in DDS-XML. This imposes some limitations on the plugin:

- In the "TRANSPORT_UNICAST QosPolicy" in the RTI Connext Core Libraries User's Manual, the **receive_port** cannot be left to the default value (0). LBED is not able to automatically compute a port number, because it doesn't know the index of the discovered *DomainParticipant*. Therefore, if you want to use this QoS policy, you need to manually specify the port number that should be used. For example:

```
<unicast>
    <value>
        <element>
            <receive_port>8080</receive_port>
            <transports>
                <element>udpv4</element>
            </transports>
        </element>
    </value>
</unicast>
```

- Only the *Connext* Builtin Transport Plugins (UDPv4, UDPv6, and SHMEM) and the *RTI Real-Time WAN Transport* (UDPv4_WAN) are supported in the "TRANSPORT_UNICAST QosPolicy" in the RTI Connext Core Libraries User's Manual and "TRANSPORT_MULTICAST QosPolicy" in the RTI Connext Core Libraries User's Manual. Defining custom aliases for them is not supported.
- Mutable QoS policies supported by LBED, such as "OWNERSHIP_STRENGTH QosPolicy" in the RTI Connext Core Libraries User's Manual (see 4.10 Supported QoS on page 43), cannot be modified at runtime. It is not possible to notify the change to the other endpoints in the system since no endpoint information is sent through the network. Endpoints rely exclusively on the configuration that is present in the static XML file.

Refer to 4.10 Supported QoS on page 43 and 4.11 Known Issues on page 46 for further information about what is and is not supported in LBED.

## 4.9 LBED Properties

Table 4.2 LBED Configuration Properties for Connext describes the name/value pairs (properties) that you can use to configure the LBED plugin. **They should be configured at the *DomainParticipant* level.**

## Table 4.2 LBED Configuration Properties for Connext

| Property Name | Property Value and Description |
|---|---|
| dds.discovery.endpoint.<br>load_plugins | Required only when the plugin is enabled using the LBED Properties (see 4.4.2 Using the LBED Plugin Properties on page 22).<br>String indicating the prefix name of the plugin that will be loaded by *Connext*.<br>Set the value to **dds.discovery.endpoint.<*string*>**, where <*string*> can be any string you want, as long as you use the same string consistently for all the properties in this table. Our example uses lbediscovery:<br>`<element>`<br>`  <name>dds.discovery.endpoint.load_plugins</name>`<br>`  <value>dds.discovery.endpoint.lbediscovery</value>`<br>`</element>` |
| dds.discovery.endpoint.<br><*string*>.library | Required only when the plugin is enabled using the LBED Properties (see 4.4.2 Using the LBED Plugin Properties on page 22).<br>The name of the dynamic library that contains the LBED plugin implementation. This library must be in the path during run time for use by *Connext*.<br>Set the value to **rtilbedisc**.<br>Example:<br>`<element>`<br>`  <name>dds.discovery.endpoint.lbediscovery.library</name>`<br>`  <value>rtilbedisc</value>`<br>`</element>` |
| dds.discovery.endpoint.<br><*string*>.create_function | Required only when the plugin is enabled using the LBED Properties (see 4.4.2 Using the LBED Plugin Properties on page 22).<br>The name of the function that will be called by *Connext* to create an instance of the LBED plugin.<br>Set the value to **DDS_LBEDiscoveryPlugin_create**.<br>Example:<br>`<element>`<br>`  <name>`<br>`  dds.discovery.endpoint.lbediscovery.create_function`<br>`  </name>`<br>`  <value>DDS_LBEDiscoveryPlugin_create</value>`<br>`</element>` |
| dds.discovery.endpoint.<br><*string*>.config_file | Optional.<br>The absolute or relative path to the file that contains the DDS-XML static definition of the endpoints that must be discovered.<br>Example:<br>`<element>`<br>`  <name>dds.discovery.endpoint.lbediscovery.config_file</name>`<br>`  <value>LBEDSubscriber.xml</value>`<br>`</element>`<br>Use this property only if:<br>• You want to use a separate file to specify the DDS-XML static definition of the endpoints instead of using the **USER_QOS_PROFILES.xml** of your application. See 4.5.2 Using a Separate XML File on page 31.<br>• You ran into the known issue "**LBED is not able to automatically load the USER_QOS_PROFILES.xml if the utilized API uses a QosProvider and the file is not placed in default locations**". See 4.11 Known Issues on page 46 for further information. |

**Table 4.2 LBED Configuration Properties for Connext**

| Property Name | Property Value and Description |
|---|---|
| dds.discovery.endpoint.<*string*>.verbosity | Optional.<br>The verbosity for the plugin, for debugging purposes.<br>• -1: Silent<br>• 0: Exceptions only (default)<br>• 1: Warnings<br>• 2 and up: Debug<br>**Example:**<br>`<element>`<br>`<name>dds.discovery.endpoint.lbediscovery.verbosity</name>`<br>`<value>1</value>`<br>`</element>`<br>Note: the LBED logging verbosity is per application. The last *DomainParticipant* using LBED and **explicitly setting this property** will apply that setting to all the *DomainParticipants* using LBED within the application. If not explicitly set, the verbosity will be left unchanged. Therefore, if no *DomainParticipant* has configured the LBED verbosity, it will be left to the default value. |
| dds.discovery.endpoint.<*string*>.property_validation_action | Optional.<br>By default, property names given in the PropertyQoSPolicy are validated to avoid using incorrect or unknown names (for example, due to a typo). This property configures the validation of the property names associated with the plugin:<br>• VALIDATION_ACTION_EXCEPTION: validate the properties. Upon failure, log errors and fail.<br>• VALIDATION_ACTION_SKIP: skip validation.<br>• VALIDATION_ACTION_WARNING: validate the properties. Upon failure, log warnings and do not fail.<br>If this property is not set, the plugin property validation behavior will be the same as that of the *DomainParticipant*, which by default is VALIDATION_ACTION_EXCEPTION. See the "Property Validation" section in the RTI Connext Core Libraries User's Manual. |

## 4.10 Supported QoS

The following lists show the QoS policies that are supported by LBED. These are the QoS policies that LBED needs to know about and that must appear in the DDS-XML file. LBED will assume the default values if any of them is not explicitly specified in the XML file. If a QoS policy that is not present in this table is specified, LBED will ignore it.

Not all of the QoS policies belong to endpoints. Some of them are specified at the *DomainParticipant*, *Topic*, *Publisher*, or *Subscriber* level. But these policies still have an impact on the configuration of the endpoint (for example, the *DomainParticipant* TRANSPORT_MULTICAST_MAPPING QoS policy may be used for determining the multicast address an endpoint should use if it has multicast enabled). Therefore, LBED also needs to know about them.

For details on these QoS, see "All QosPolicies" in the RTI Connext Core Libraries User's Manual.

**DomainParticipantQos**

- TransportMulticastMapping
- The following field only in the DomainParticipantResourceLimits QoS policy:
  - **channel_seq_max_length**
- TransportBuiltin
- The following field only in the WireProtocol QoS policy:
  - **rtps_well_known_ports**

**TopicQos**

- TopicData

**PublisherQos/SubscriberQos**

- GroupData
- Partition
- Presentation

**DataWriterQos**

- Deadline
- DestinationOrder
- Durability
- DurabilityService
- LatencyBudget
- Lifespan
- Liveliness
- Ownership
- OwnershipStrength
- Property

  In the Simple Discovery Protocol (SDP), properties are not propagated by default during discovery, so you likely won't need to use the Property QoS policy in LBED. However, if you do

want your endpoints' properties to be known by other endpoints in LBED, you can use the Property QoS policy to do that.

- The following fields only in the DataWriterProtocol QoS policy:
  - **disable_positive_acks**
  - **rtps_object_id**
  - **virtual_guid**
- Reliability
- Service
- TransportSelection
- TransportUnicast
- UserData

**DataReaderQos**

- Deadline
- DestinationOrder
- Durability
- LatencyBudget
- Liveliness
- TransportMulticast
- Ownership
- Property

  In the Simple Discovery Protocol (SDP), properties are not propagated by default during discovery, so you likely won't need to use the Property QoS policy in LBED. However, if you do want your endpoints' properties to be known by other endpoints in LBED, you can use the Property QoS policy to do that.

- The following fields only in the DataWriterProtocol QoS policy:
  - **disable_positive_acks**
  - **rtps_object_id**
  - **virtual_guid**
- Reliability

- Service
- TimeBasedFilter
- TransportSelection
- TransportUnicast
- UserData

## 4.11 Known Issues

### 4.11.1 LBED not able to automatically load USER_QOS_PROFILES.xml if utilized API uses QosProvider and the file is not placed in default locations

If you are using an API that utilizes a QosProvider for loading XML QoS profiles (e.g., for Modern C++ or C#), the LBED plugin won't be able to automatically load the XML file with the endpoint configurations if both of the following are true:

- You defined the LBED information in **USER_QOS_PROFILES.xml**, which is neither located in the working directory nor referenced in the NDDS_QOS_PROFILES environment variable.
- The content of **USER_QOS_PROFILES.xml** is not specified in the **<NDDSHOME>/resource/xml/NDDS_QOS_PROFILES.xml** file.

See "How to Load XML-Specified QoS Settings" in the RTI Connext Core Libraries User's Manual for more information on where *Connext* looks for XML QoS profiles. The LBED plugin won't be able to automatically load the XML file with the endpoint configurations if the XML content is not in the expected locations. This is because the LBED plugin relies on the singleton DomainParticipantFactory for loading the profiles, but the DomainParticipantFactory is not used in APIs that use a QosProvider.

As a workaround to make LBED load the file, set the **config_file** property in your **USER_QOS_PROFILES.xml** file with the path to itself. For example, if the **USER_QOS_PROFILE.xml** file is located in **/foo/bar/**, the <domain_participant_qos> of your participant enabling LBED should look like this:

```
<!-- USER_QOS_PROFILES.xml -->
<qos_profile name="...">
    <domain_participant_qos>
        <discovery_config>
            <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
        </discovery_config>
        <property>
            <value>
                <element>
                    <name>dds.discovery.endpoint.lbediscovery.config_file</name>
                    <value>/foo/bar/USER_QOS_PROFILES.xml</value>
                </element>
```

```
                ...
            </value>
        </property>
        ...
    </domain_participant_qos>
    ...
</qos_profile>
```

# Chapter 5 Limited Bandwidth RTPS Transport Plugin

The Real-Time Publish Subscribe (RTPS) communication protocol is used by the Data Distribution Service (DDS) interoperability protocol. *Connext* uses RTPS packages to send data over the network. The Limited Bandwidth RTPS (LBRTPS) transport plugin reduces the size of the message headers in the RTPS packages sent over the network. The message headers are reduced by eliminating some fields and making other fields smaller.

This chapter provides a brief overview of how RTPS messages are structured and describes how to configure the LBRTPS transport plugin. More information about RTPS can be found in the OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5. See also the "Wire Protocol Compatibility" section in the RTI Connext Core Libraries Release Notes for further clarification.

> You must link with the *dynamic* version of the *Connext* libraries. See the RTI Connext Core Libraries Platform Notes for details.

## 5.1 Understanding the RTPS Message Header

The overall structure of an RTPS *Message* consists of a fixed-size RTPS *Header* followed by a variable number of RTPS *Submessage* parts. Each *Submessage* consists of a *SubmessageHeader* and a variable number of *SubmessageElements*. The RTPS header must appear at the beginning of every message.

The *Header* contains these fields:

- **protocol:** Identifies the message as an RTPS message.

  Representation: 4 bytes

```
0.........8........16........24........32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|   'R'   |   'T'   |   'P'   |   'S'   |
+---------+---------+---------+---------+
```

- **version:** Identifies the version of the RTPS protocol. .

  Representation: 2 bytes. This release uses version {2, 1}.
- **vendorId:** Indicates the vendor that provides the implementation of the RTPS protocol.

  Representation: 2 bytes. The RTI vendor identifier is {1, 1}.
- **guidPrefix:** Defines a default prefix to use for all GUIDs that appear in the message.

  Representation: 12 bytes: 4 bytes for the host identifier, 4 bytes for the application identifier and 4 bytes for the instance identifier.

Figure 5.1: RTPS Message Structure

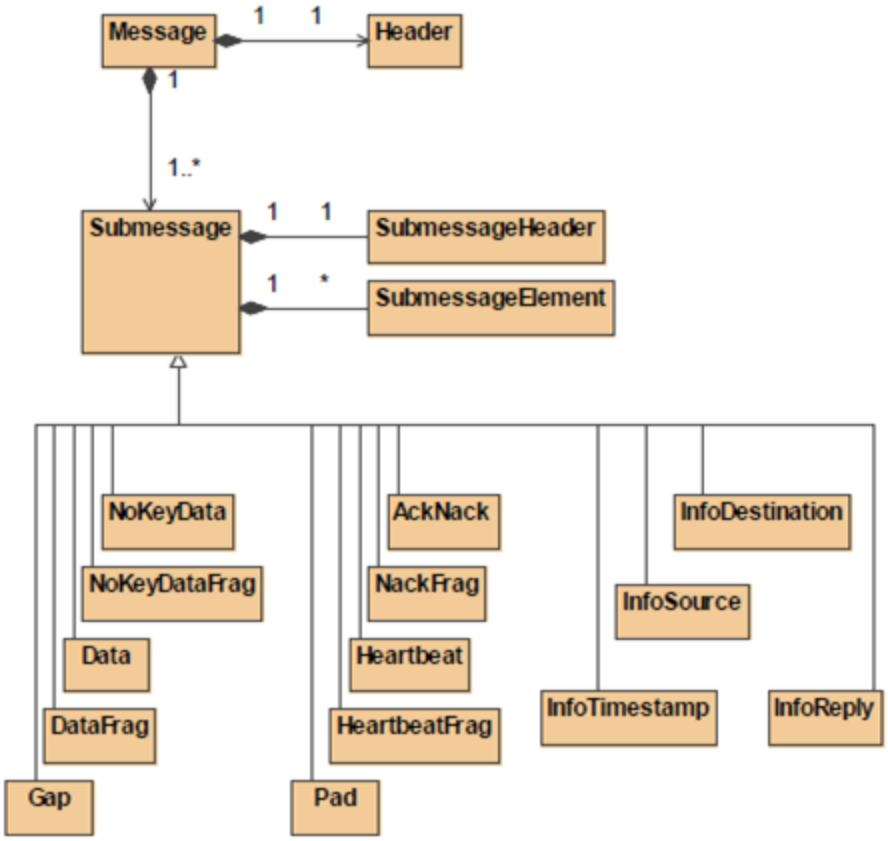
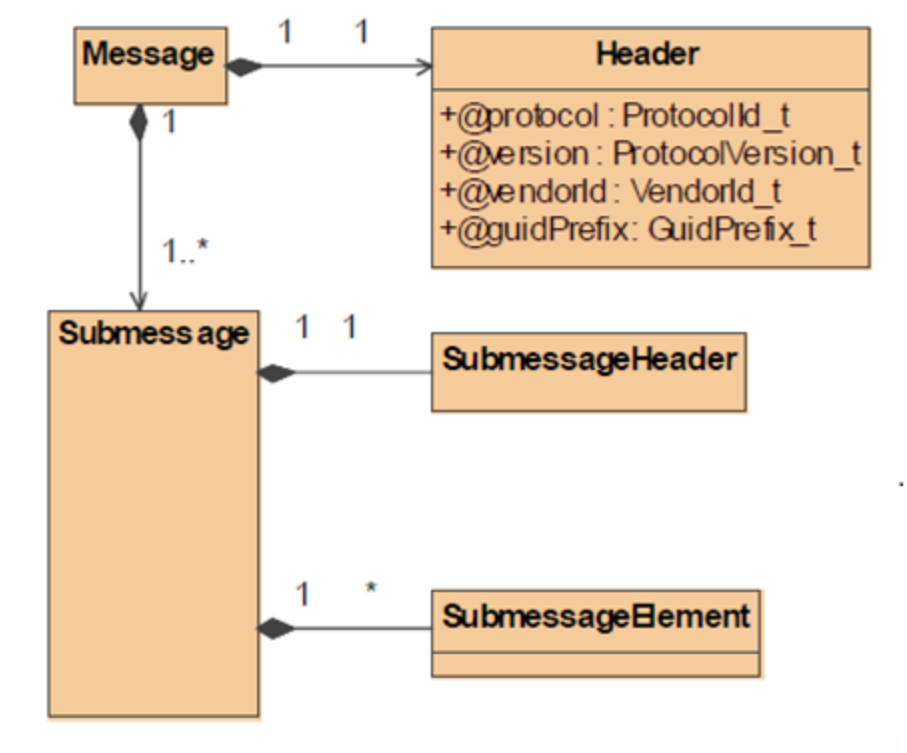


Figure 5.2: RTPS Message Header Structure




The LBRTPS transport reduces the RTPS message headers by eliminating the **protocol**, **version**, and **vendorId** fields in the RTPS *Header* structure.

### 5.1.1 Submessage Structure

Each RTPS message consists of a variable number of RTPS submessages. All RTPS submessages have the same structure; they start with a *SubmessageHeader,* followed by a concatenation of *SubmessageElement* parts. The *SubmessageHeader* identifies the kind of submessage and the optional elements within that submessage.

The *SubmessageHeader* contains these fields:

- **submessageId:** A 1-byte field that identifies the kind of submessage.
- **flags:** A 1-byte field that identifies the endianness used to encapsulate the submessage, the presence of optional elements within the submessage, and possibly modifies the interpretation of the submessage.
- **submessageLength:** A 2-byte field that indicates the length of the submessage. Given that an RTPS message consists of a concatenation of submessages, the submessage length can be used to skip to the next submessage.

DATA and DATA_FRAG submessages contain a field called **extraflags**. It provides space (2 bytes) for an additional 16 bits of flags beyond the 8 bits provided in the *SubmessageHeader*.

ACKNACK, HEARTBEAT, GAP, ACKNACK_FRAG, HEARTBEAT_FRAG, DATA, and DATA_FRAG submessages contain a reader entity identifier and a writer entity identifier (*readerId* field and *writerId* field). The representation is 4 bytes. These submessages also contain an 8-byte field for a sequence number.

## 5.2 Configuring the LBRTPS Transport

The LBRTPS transport must be created using the *Connext* Transport API; for more information, please see *Transport Plugins*, in the RTI Connext Core Libraries User's Manual.

Before we describe the name/value pairs that can be used in the Property QoS policy (see the "PROPERTY QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual) to configure the LBRTPS transport, let's review an example. The first step is to disable the builtin transports by configuring the TransportBuiltin QoS policy with the mask **MASK_NONE**. Then the name/value pairs in the Property QoS policy are set up to load and configure the LBRTPS transport plugin.

```
<qos_library name="Property_Library">
    <qos_profile name="Property_Profile">
        <domain_participant_qos>
        ...
        <transport_builtin>
        <mask>MASK_NONE</mask>
        </transport_builtin>
        <property>
        <value>
                <element>
            <name>dds.transport.load_plugins</name>
            <value>dds.transport.lbrtps</value>
            </element>
            <element>
                <name>dds.transport.lbrtps.library</name>
            <value>rtilbrtps</value>
            </element>
            <element>
            <name>dds.transport.lbrtps.create_function</name>
                <value>LBRTPS_Transport_create_plugin</value>
            </element>
            <element>
                <name>dds.transport.lbrtps.subtransport</name>
            <value>UDPv4</value>
            </element>
            <element>
            <name>dds.transport.lbrtps.aliases</name>
            <value>lbrtps.udpv4</value>
            </element>
            <element>
                <name>
```

```
                        dds.transport.lbrtps.UDPv4.multicast_enabled
                    </name>
            <value>1</value>
            </element>
            <element>
            <name>
                    dds.transport.lbrtps.rtps_header.eliminate_protocol
            </name>
            <value>true</value>
            </element>
            <element>
            <name>
                        dds.transport.lbrtps.rtps_header.eliminate_version
            </name>
            <value>true</value>
            </element>
            <element>
                <name>
                        dds.transport.lbrtps.rtps_header.eliminate_vendorId
                </name>
            <value>true</value>
            </element>
            ...
        </value>
        </property>
        ...
    </domain_participant_qos>
    </qos_profile>
</qos_library>
```

When using the above example, *Connext* will load the LBRTPS transport plugin from the library **lbrtps.dll** and call the function **LBRTPS_Transport_create_plugin()**, which will create the LBRTPS transport. The LBRTPS transport is designed to work over the *Connext* UDPv4 transport. The LBRTPS transport is registered in *Connext* with the participant that uses this QoS profile.

> *Connext* does not assign initial peers to the LBRTPS transport plugin. You can set the initial peers with NDDS_DISCOVERY_PEERS, as described in the *Discovery Overview* chapter in the RTI Connext Core Libraries User's Manual. The LBRTPS transport plugin example uses the NDDS_DISCOVERY_PEERS file to set the multicast address lbrtps://239.255.0.1.

Table 5.1 Configuration Properties for LBRTPS Plugin describes the name/value pairs that you can use to configure the LBRTPS transport.

## Table 5.1 Configuration Properties for LBRTPS Plugin

| Property Name | Property Value and Description |
|---|---|
| dds.transport.<br>load_plugins | Required.<br>Comma-separated strings indicating the prefix names of all plugins that will be loaded by *Connext*.<br>Must be set to dds.transport.lbrtps.<br>Example:<br>`<element>`<br>`<name>dds.transport.load_plugins</name>`<br>`<value>dds.transport.lbrtps</value>`<br>`</element>` |
| dds.transport.lbrtps.<br>subtransport | Required.<br>Name of the plugin to be loaded by the LBRTPS transport. The LBRTPS transport will work over this loaded plugin. The value can be UDPv4, zrtps, or a user-specified string; see 5.2.1 Configuring the LBRTPS Transport Plugin's 'Subtransport' Property on page 56. |
| dds.transport.lbrtps.<br>library | Required.<br>The name of the dynamic library that contains the LBRTPS transport plugin implementation. This library must be in the path during run time for use by *Connext*.<br>Example:<br>`<element>`<br>`<name>dds.transport.lbrtps.library</name>`<br>`<value>rtilbrtps</value>`<br>`</element>` |
| dds.transport.lbrtps.<br>create_function | Required.<br>The name of the function that will be called by *Connext* to create an instance of the LBRTPS transport. The function must have the prototype of NDDS_Transport_create_plugin.<br>Must be set to LBRTPS_Transport_create_plugin.<br>Example:<br>`<element>`<br>`<name>dds.transport.lbrtps.create_function</name>`<br>`<value>LBRTPS_Transport_create_plugin</value>`<br>`</element>` |
| dds.transport.lbrtps.<br>aliases | Required.<br>Aliases used to register the LBRTPS transport plugin with the *DomainParticipant*. The transport must have been created by the dds.transport.lbrtps.create_function. Aliases should be specified as comma-separated strings, with each comma delimiting an alias.<br>An example alias for the LBRTPS transport, working over a *Connext* UDPv4 transport: lbrtps.udpv4.<br>Example:<br>`<element>`<br>`<name>dds.transport.lbrtps.aliases</name>`<br>`<value>lbrtps.udpv4</value>`<br>`</element>` |
| The following properties are optional and appear in alphabetical order. | |
| dds.transport.lbrtps.<br>rtps_header.<br>eliminate_protocol | Whether or not to eliminate the 4-byte protocol field in the RTPS header.<br>Must be a boolean value: true or false.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.transport.lbrtps.rtps_header.eliminate_protocol`<br>`</name>`<br>`<value>true</value>`<br>`</element>` |

## Table 5.1 Configuration Properties for LBRTPS Plugin

| Property Name | Property Value and Description |
|---|---|
| dds.transport.lbrtps.<br>rtps_header.<br>eliminate_vendorId | Whether or not to eliminate the 2-byte vendorId field in the RTPS header.<br>Must be a boolean value: true or false.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.transport.lbrtps.rtps_header.eliminate_vendorId`<br>`</name>`<br>`<value>true</value>`<br>`</element>` |
| dds.transport.lbrtps.<br>rtps_header.<br>eliminate_version | Whether or not to eliminate the 2-byte version field in the RTPS header.<br>Must be a boolean value: true or false.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.transport.lbrtps.rtps_header.eliminate_version`<br>`</name>`<br>`<value>true</value>`<br>`</element>` |
| dds.transport.lbrtps.<br>rtps_header.<br>reduce_guidPrefix | The reduce_guidPrefix field is comprised of 12 bytes that represent 3 different fields of 4 bytes each: hostId, appId and instanceId.<br>See 5.2.2 Configuring the LBRTPS Transport Plugin's 'reduce_guidPrefix' Property on page 59. |
| dds.transport.lbrtps.<br>submessage_header.<br>combine_<br>submessageId_<br>with_flags | The first two fields in the SubmessageHeader do not use all the bits. The first field, submessageId, only uses 5 bits. The second field, flags, only uses 3 bits. By setting this property to true, these two fields can be packed into a single byte.<br>Must be set to a boolean value: true or false.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.transport.lbrtps.submessage_header.combine_submessageId_with_flags`<br>`</name>`<br>`<value>true</value>`<br>`</element>` |

## Table 5.1 Configuration Properties for LBRTPS Plugin

| Property Name | Property Value and Description |
|---|---|
| dds.transport.lbrtps.<br>submessages.<br>reduce_entitiesId | Many submessage kinds include two fields: readerId and writerId.<br>In the RTPS protocol specification, these fields are mapped to structure:<br>`struct { octet[3] entityKey; octet entityKind; } EntityId_t;`<br>As you can see, 4 bytes are allocated for each entity: 3 for the entityKey and 1 for the entityKind. However, you may be able to reduce the size of these fields if you know ahead of time how many *DataReaders* and *DataWriters* you will have.<br>If you will have no more than 2048 *DataReaders* and 2048 *DataWriters*, you can reduce the size of each of these fields from four to two bytes. And if you will have no more than 8 *DataReaders* and 8 *DataWriters*, you can reduce each field to only one byte. The math involved is explained below.<br>Five bits are always needed for the entityKind. If you have no more than 2048 *DataReaders* and 2048 *DataWriters*, their entityKeys can be 0-2047, which will fit in 11 bits. Thus you only need 2 bytes in this case: 5 bits for entityKind + 11 bits for entityKey = 16 bits = 2 bytes.<br>Suppose you have no more than 8 *DataReaders* and 8 *DataWriters*. In this case, you still need 5 bits for the entityKind, but only 3 bits to hold entityKeys 0-7. So you would only need 8 bits: 5 bits for entityKind + 3 bits for entityKey = 8 bits = 1 byte.<br>This property's value is expressed as 2 comma-separated integers between 8 and 32, to specify the number of bits to use for the readerId and writerId.<br>For example, to reduce both readerId and writerId to 12 bits each:<br>`<element> <name>dds.transport.lbrtps.submessages.reduce_entitiesId</name> <value>12,12</value> </element>` |
| dds.transport.lbrtps.<br>submessages.<br>reduce_<br>sequenceNumber | Some submessage kinds keep track of the number of submessages received by using an 8-byte sequence number field. You can reduce the number of bytes used for the sequence number by using this property.<br>The value must be an integer between 16 and 64 that specifies the desired size of the sequence number, in bits.<br>Example:<br>`<element> <name> dds.transport.lbrtps.submessages.reduce_sequenceNumber </name> <value>32</value> </element>` |

**Table 5.1 Configuration Properties for LBRTPS Plugin**

| Property Name | Property Value and Description |
|---|---|
| dds.transport.lbrtps.verbosity | The verbosity for the plugin, for debugging purposes.<br>• -1: Silent<br>• 0: Exceptions only (default)<br>• 1: Warnings<br>• 2 and up: Debug<br><br>Example:<br>`<element>`<br>`<name>dds.transport.lbrtps.verbosity>/name>`<br>`<value>0</value>`<br>`</element>`<br><br>Note: the LBRTPS logging verbosity is per application. The last *DomainParticipant* using LBRTPS and **explicitly setting this property** will apply that setting to all the *DomainParticipants* using LBRTPS within the application. If not explicitly set, the verbosity will be left unchanged. Therefore, if no *DomainParticipant* has configured the LBRTPS verbosity, it will be left to the default value. |
| dds.transport.lbrtps.property_validation_action | Optional.<br>By default, property names given in the PropertyQoSPolicy are validated to avoid using incorrect or unknown names (for example, due to a typo). This property configures the validation of the property names associated with the plugin:<br>• VALIDATION_ACTION_EXCEPTION: validate the properties. Upon failure, log errors and fail.<br>• VALIDATION_ACTION_SKIP: skip validation.<br>• VALIDATION_ACTION_WARNING: validate the properties. Upon failure, log warnings and do not fail.<br><br>If this property is not set, the plugin property validation behavior will be the same as that of the *DomainParticipant*, which by default is VALIDATION_ACTION_EXCEPTION. See the "Property Validation" section in the RTI Connext Core Libraries User's Manual. |

## 5.2.1 Configuring the LBRTPS Transport Plugin's 'Subtransport' Property

The required property, **dds.transport.lbrtps.subtransport**, specifies the plugin to be loaded by the LBRTPS transport. The value can be **UDPv4**, **zrtps**, or a user-specified value, as described in the following sections. Once you set the value for a subtransport, the names of all the properties for that subtransport should be in the form **dds.transport.lbrtps.<*subtransport*>.<*property*>**.

### 5.2.1.1 Using UDPv4 as a Subtransport

To load the *Connext* UDPv4 built-in transport, use the value **UDPv4**. If you want the UDPv4 transport to be created with multicast support, also set **dds.transport.lbrtps.UDPv4.multicast_enabled** to 1, as seen in the example below.

To use the UDPv4 transport and enable multicast:

```
<element>
    <name>dds.transport.lbrtps.subtransport</name>
    <value>UDPv4</value>
</element>
<element>
```

```
    <name>dds.transport.lbrtps.UDPv4.multicast_enabled</name>
    <value>1</value>
</element>
```

#### 5.2.1.2 Using ZRTPS as a Subtransport

In Chapter 6 Compression Real-Time Publish Subscribe Transport Plugin on page 61, you will learn about the Compression Real-Time Publish Subscribe Transport Plugin (ZRTPS). You can use the LBRTPS and ZRTPS transport plugins together, as long as you meet the following three requirements. By using this combination of transport plugins, the LBRTPS transport plugin will reduce the RTPS headers, then the ZRTPS transport plugin will compress the RTPS package.

The LBRTPS transport plugin properties must appear in the XML QoS profile *before* those for the ZRTPS transport plugin. (See the example below.)

As mentioned in Chapter 1 Introduction on page 1, the plugins are executed in the order in which they appear in the XML file. So having the LBRTPS properties appear in the file before the ZRTPS properties is important because once the message is compressed by the ZRTPS transport plugin, the header is no longer recognizable by the LBRTPS transport plugin as an RTPS header.

The ZRTPS transport plugin must be configured as a subtransport of the LBRTPS transport plugin using the value **zrtps**, as seen here:

```
<element>
    <name>dds.transport.lbrtps.subtransport</name>
    <value>zrtps</value>
</element>
```

You will also need to set two additional properties**:**

- **dds.transport.lbrtps.zrtps.library**
- **dds.transport.lbrtps.zrtps.create_function**

```
<element>
    <name>dds.transport.lbrtps.zrtps.library</name>
    <value>rtizrtps</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.create_function</name>
    <value>ZRTPS_Transport_create_plugin</value>
</element>
```

The LBRTPS and ZRTPS transport plugins cannot both use UDPv4 as a subtransport because of port conflict issues.

The following example configures the LBRTPS transport plugin with ZRTPS as a subtransport.

```
<!-- LBRTPS -->
<element>
   <name>dds.transport.load_plugins</name>
   <value>dds.transport.lbrtps</value>
```

```
</element>
<element>
   <name>dds.transport.lbrtps.library</name>
   <value>rtilbrtps</value>
</element>
<element>
   <name>dds.transport.lbrtps.create_function</name>
   <value>LBRTPS_Transport_create_plugin</value>
</element>
<element>
   <name>dds.transport.lbrtps.aliases</name>
   <value>lbrtps.zrtps</value>
</element>
<element>
   <name>dds.transport.lbrtps.subtransport</name>
   <value>zrtps</value>
</element>
<element>
   <name>dds.transport.lbrtps.verbosity</name>
   <value>2</value>
</element>
<!-- ZRTPS-->
<element>
   <name>dds.transport.lbrtps.zrtps.library</name>
   <value>rtizrtps</value>
</element>
<element>
   <name>dds.transport.lbrtps.zrtps.create_function</name>
   <value>ZRTPS_Transport_create_plugin</value>
</element>
<element>
   <name>dds.transport.lbrtps.zrtps.subtransport</name>
   <value>UDPv4</value>
</element>
<element>
   <name>dds.transport.lbrtps.zrtps.UDPv4.multicast_enabled</name>
   <value>1</value>
</element>
<element>
   <name>dds.transport.lbrtps.zrtps.compression_library</name>
   <value>AUTOMATIC_COMPRESSION</value>
</element>
<element>
   <name>dds.transport.lbrtps.zrtps.compression_level</name>
   <value>9</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.verbosity</name>
    <value>0</value>
</element>
```

#### 5.2.1.3 Using a User-Specified Subtransport

To load a user-provided transport plugin, provide a value for **dds.transport.lbrtps.subtransport**, and then use that same value like a prefix to define these two additional properties:

- **dds.transport.lbrtps.*prefix*.library**
- **dds.transport.lbrtps.*prefix*.create_function**

For example, to specify the value, **testplugin**:

```
<element>
    <name>dds.transport.lbrtps.subtransport</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.lbrtps.testplugin.library</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.lbrtps.testplugin.create_function</name>
    <value>TestPlugin_Transport_create_function</value>
</element>
```

### 5.2.2 Configuring the LBRTPS Transport Plugin's 'reduce_guidPrefix' Property

The **dds.transport.lbrtps.rtps_header.reduce_guidPrefix** property is comprised of 12 bytes that represent three different 4-byte fields:

- **hostId**: A machine/OS-specific host identifier, unique in the domain. If you know the number of machines in the network ahead of time, the size of the **hostId** field can be reduced. For example, if only two machines are connected in the network, then this field can be reduced to two bits (the identifiers would be '1' and '2', in binary '01' and '11'). The range of host IDs that can be used in the WireProtocolQosPolicy is therefore limited by this value.
- **appId**: A participant-specific identifier, unique within the scope of the **hostId**. If you know the number of participants in the domain ahead of time, the size of the **appId** field can be reduced. For example, if only four participants are in the domain, then this field can be reduced to three bits. The range of **appId**'s that can be used in the WireProtocolQosPolicy is therefore limited by this value.
- **instanceId**: An instance-specific identifier of the *DomainParticipant* that, together with the **appId**, is unique within the scope of the **hostId**. This identifier is increased each time the participant is recreated in the application. Since most applications create only one *DomainParticipant*, this field can usually be eliminated—in which case a value of 1 is assumed.

The **reduce_guidPrefix** value is expressed as three comma-separated integers between 0 and 32. The values specify the number of bits to use for each identifier (**hostId**, **appId** and **instanceId**).

In the example below, the **hostId** will use 5 bits (because in our example network there will be 32 machines), the **appId** field will use 6 bits (because there will be 64 participants in the domain), and the **instanceId** field will be eliminated. So 11 bits will be used. Then the LBRTPS transport will reduce the **guidPrefix** field 11 bits into 2 bytes.

```
<element>
    <name>
    dds.transport.lbrtps.rtps_header.reduce_guidPrefix
    </name>
    <value>5,6,0</value>
</element>
```

**Important Notes:**

1. An integral number of bytes is sent by the plugin for the GUID prefix. So the plugin will round up to the next byte when the total number of bits for the IDs add up to less than an integral number of bytes. For example, a **reduce_guidPrefix** value of 3,2,0 requires 5 bits, so 1 byte will be sent; a value of 4,4,2 requires 10 bits so 2 bytes will be sent; etc.
2. Setting any of the **reduce_guidPrefix** fields to 0 will behave as expected on the sending side (no bits are used to send that ID) but on the receiving side, the plugin will use/assume an ID value of 1 for any IDs that were not sent. Thus, when setting 0 bits for an ID field in the **reduce_guidPrefix**, you must use the WireProtocolQosPolicy to set the corresponding ID to a value of 1 in the local participant.
3. When using this property to reduce the number of bits used to encode an ID, the actual ID (host, app, instance) should be a value that can be represented by the reduced bits. If a full ID is used, aliasing of two full ID values to the same reduced ID value may occur since a bit mask is used to convert a full ID to the reduced ID. This could lead to two different participants having the same reduced GUID prefix. For example, using '2,0,0' for the **reduce_guidPrefix** property (2 bits to encode the **host_id** and no bits for the app ID or instance ID) and setting the **rtps_host_id** in the WireProtocolQosPolicy to 3 (0011 in binary) for one participant and 7 (0111 binary) for another participant will cause the reduced **rtps_host_id**'s of both participants to be the same value of 3 (0011 binary). In this situation, discovery will not complete.
4. A **reduce_guidPrefix** value of '0,0,0' is not valid. As noted in point 2 above, a **reduce_guidPrefix** of '0,0,0' will be interpreted as a GUID prefix of host ID = 1, app ID = 1 and instance ID = 1. Unfortunately, this would lead to all participants having the same GUID prefix, since a GUID prefix of host ID = 0, app ID = 0 and instance ID = 0 is not allowed. As a side effect, the GUID prefix cannot be reduced to 0 bytes (1-byte minimum used to send the GUID prefix).
5. When setting the WireProtocolQosPolicy of a participant, a value of 0 for an ID is equivalent to setting the value to DDS_RTPS_AUTO_ID. So when using 0 for an ID, DDS will automatically set the value of the ID to some value other than 0.

# Chapter 6 Compression Real-Time Publish Subscribe Transport Plugin

Real-Time Publish Subscribe (RTPS) is the communication protocol used by the Data Distribution Service (DDS) interoperability protocol. *Connext* uses RTPS packages to send data over the network.

The Compression Real-Time Publish Subscribe (ZRTPS) transport plugin reduces the size of the RTPS packages sent over the network by *Connext*.

You can configure the ZRTPS transport plugin to use one of the following algorithms to compress all RTPS packages:

- **Zlib**: This compression library is an abstraction of the DEFLATE compression algorithm used in the gzip file compression program. This is free software, distributed under the ZLIB license.
    - **Windows users: zlib1.dll** is included in the **<NDDSHOME>\lib\<*architecture*>** directory (where NDDSHOME is described in Chapter 2 Paths Mentioned in Documentation on page 5 and <*architecture*> is one of the supported architecture strings listed in the *RTI Connext Core Libraries Release Notes*).
    - **Linux users: libzlib1.so** is likely already installed on your system. If you need to get a Zlib library, please see http://zlib.net for information on how to obtain this library for your platform.
- **Bzip2**: This compression library contains the bzip2 compression algorithm. This algorithm is a lossless data compression algorithm. bzip2 compresses most files more effectively than the older LZW and Deflate compression algorithms, but is considerably slower (~12 times vs. Deflate on typical data). This is free software, distributed under the BSD license.

- **External compression library**: You can add your own compression library and configure the ZRTPS transport plugin to use it.

The transport also supports an automatic mode, which will select from the available algorithms the one that results in the smallest compressed package. While this automatic mode assures the best size reduction, it is slower and uses more memory.

The ZRTPS transport plugin can apply different compression algorithms, depending on each RTPS package's size (small, medium, and large—these sizes are also user-configurable).

The ZRTPS transport works over another transport. It cannot send data over a network by itself. It can work over the *Connext* UDPv4 transport, or a custom transport created using the *Connext* Transports API. You can configure *Connext* to use the ZRTPS transport via the QoS profiles XML; see 6.2 Configuring the ZRTPS Transport below.

> You must link with the **dynamic** version of the *Connext* core libraries. See the *RTI Connext Core Libraries Platform Notes* for details.

## 6.1 Transport-Related Limitations

The following are known transport-interaction limitations when using the ZRTPS transport plugin:

- Using LBRTPS *and* ZRTPS: See 5.2.1.2 Using ZRTPS as a Subtransport on page 57.
- Neither Shared Memory (SHMEM) nor UDPv6 may be used as subtransports.
- The UDPv4 transport may not be used simultaneously as a transport and a ZRTPS subtransport.

## 6.2 Configuring the ZRTPS Transport

This section describes how to configure the properties for the ZRTPS Transport plugin in the XML QoS Profile file used by *Connext* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQosPolicy for your application's *DomainParticipant*. (See the "PROPERTY QosPolicy (DDS Extension)" in the RTI Connext Core Libraries User's Manual.)

Before we describe the name/value pairs that can be used in the Property QoS policy to configure the ZRTPS transport, let's review an example. The first step is to disable the built-in transports by configuring the TransportBuiltin QoS policy with the mask **MASK_NONE**. Then the name/value pairs in the Property QoS policy are set up to load and configure the ZRTPS transport plugin**.**

```
<qos_library name="Property_Library">
    <qos_profile name="Property_Profile">
    <domain_participant_qos>
        ...
            <transport_builtin>
                <mask>MASK_NONE</mask>
            </transport_builtin>
```

```
            <property>
                <value>
                <element>
                    <name>dds.transport.load_plugins</name>
                    <value>dds.transport.zrtps</value>
                </element>
                <element>
                    <name>dds.transport.zrtps.library</name>
                    <value>rtizrtps</value>
                </element>
                <element>
                    <name>dds.transport.zrtps.create_function</name>
                    <value>ZRTPS_Transport_create_plugin</value>
                    </element>
                <element>
                    <name>dds.transport.zrtps.subtransport</name>
                    <value>UDPv4</value>
                </element>
                    <element>
                    <name>dds.transport.zrtps.aliases</name>
                        <value>zrtps.udpv4</value>
                </element>
                    <element>
                        <name>dds.transport.zrtps.UDPv4.multicast_enabled</name>
                    <value>1</value>
                </element>
                <element>
                    <name>dds.transport.zrtps.compression_library</name>
                    <value>AUTOMATIC_COMPRESSION</value>
                </element>
                <element>
                        <name>dds.transport.zrtps.compression_level</name>
                    <value>9</value>
                </element>
                ...
                </value>
            </property>
            ...
        </domain_participant_qos>
    </qos_profile>
</qos_library>
```

When using the above QoS profile, *Connext* will load the ZRTPS transport plugin from the library, rtizrtps.dll on Windows systems, or rtizrtps.so on Linux systems and call the function **ZRTPS_Transport_create_plugin()** to create the ZRTPS transport.

The ZRTPS transport is designed to work over the *Connext* UDPv4 transport. The automatic mode described previously is set in the ZRTPS transport and the compression level for all compression algorithms is set to 9. The ZRTPS transport will be registered in *Connext* with the participant that uses this QoS profile.

*Connext* does not assign initial peers to the ZRTPS transport plugin. You can set the initial peers with NDDS_DISCOVERY_PEERS, as described in the chapter on Discovery in the *RTI Connext Core Libraries User's Manual*. The ZRTPS transport plugin example uses the NDDS_DISCOVERY_PEERS file to set the multicast address, zrtps.udpv4://239.255.0.1.

### Table 6.1 Configuration Properties for ZRTPS Plugin

| Property Name | Property Value and Description |
|---|---|
| dds.transport.load_plugins | Required.<br>Comma-separated strings indicating the prefix names of all plugins to be loaded by *Connext*.<br>Set the value to **dds.transport.zrtps**.<br>Example:<br>`<element>`<br>`<name>dds.transport.load_plugins</name>`<br>`<value>dds.transport.zrtps</value>`<br>`</element>` |
| dds.transport.zrtps.library | Required.<br>The name of the dynamic library that contains the ZRTPS transport plugin implementation. This library must be in the path during run time for use by *Connext*.<br>The value must be **rtizrtps**.<br>Example:<br>`<element>`<br>`<name>dds.transport.zrtps.library</name>`<br>`<value>rtizrtps</value>`<br>`</element>` |
| dds.transport.zrtps.create_function | Required.<br>The name of the function that will be called by *Connext* to create an instance of the ZRPTS transport. The function must have the prototype **NDDS_Transport_create_plugin**.<br>Must be set to **ZRTPS_Transport_create_plugin**.<br>Example:<br>`<element>`<br>`<name>dds.transport.zrtps.create_plugin</name>`<br>`<value>ZRTPS_Transport_create_plugin</value>`<br>`</element>` |
| dds.transport.zrtps.subtransport | Required.<br>Name of the plugin to be loaded by the ZRTPS transport. The ZRTPS transport will work over this loaded plugin. The value can be **UDPv4** or a user-specified string; see 6.2.1 Configuring the ZRTPS Transport Plugin's 'Subtransport' Property on page 67.<br>Example:<br>`<element>`<br>`<name>dds.transport.zrtps.subtransport</name>`<br>`<value>UDPv4</value>`<br>`</element>` |
| dds.transport.zrtps.aliases | Required.<br>Aliases used to register the ZRTPS transport plugin with the *DomainParticipant*. The transport must have been created by the dds.transport.zrtps.create_function. Aliases should be specified as a comma-separated string, with each comma delimiting an alias.<br>Example:<br>`<element>`<br>`<name>dds.transport.zrtps.aliases</name>`<br>`<value>zrtps.udpv4</value>`<br>`</element>` |

## Table 6.1 Configuration Properties for ZRTPS Plugin

| Property Name | Property Value and Description |
|---|---|
| *The following properties are optional.* | |
| dds.transport.zrtps.compression_level | Defines the compression level that the compression algorithm will use for all RTPS packages. In automatic mode (see dds.transport.zrtps.compression_library), all compression algorithms will use this level.<br>The value must be an integer between 1 and 9 (inclusive). A lower value will result in less time spent doing the compression; a higher value may result in a higher compression percentage (smaller compressed output).<br>Example:<br>`<element>`<br>`  <name>dds.transport.zrtps.compression_level</name>`<br>`  <value>9</value>`<br>`</element>` |
| dds.transport.zrtps.compression_level.small_packets<br>dds.transport.zrtps.compression_level.medium_packets<br>dds.transport.zrtps.compression_level.large_packets | Defines the compression level that the compression algorithm will use for small/medium/large RTPS packages. In automatic mode (see dds.transport.zrtps.compression_library), all compression algorithms will use this level.<br>The value must be an integer between 1 and 9 (inclusive). A lower value means more speed compression; a higher value means more size reduction.<br>Example:<br>`<element>`<br>`  <name>`<br>`  dds.transport.zrtps.compression_level.small_packets`<br>`  </name>`<br>`  <value>9</value>`<br>`</element>`<br>`<element>`<br>`  <name>`<br>`  dds.transport.zrtps.compression_level.medium_packets`<br>`  </name>`<br>`  <value>9</value>`<br>`</element>`<br>`<element>`<br>`  <name>`<br>`  dds.transport.zrtps.compression_level.large_packets`<br>`  </name>`<br>`  <value>9</value>`<br>`</element>` |
| dds.transport.zrtps.compression_library | Specifies the compression algorithm to be used by the ZRTPS transport for all RTPS packages. These compression algorithms are in external libraries. The ZRTPS transport only uses the libraries whose algorithms will be used. The required libraries must be in the path during run time so the ZRTPS transport can find them.<br>There are several compression algorithms that can be used to compress RTPS packages:<br>• ZLIB_COMPRESSION: Use the Zlib algorithm from the library **zlib1.dll**.<br>• BZIP2_COMPRESSION: Use the Bzip2 algorithm from the library **bzip2.dll**.<br>• EXTERNAL_COMPRESSION: Use an external compression library defined in the property dds.transport.zrtps.external_library. See 6.2.2 Configuring the External Compression Library on page 68.<br>• AUTOMATIC_COMPRESSION: Compress RTPS packages with all previous compression algorithms and send the smallest package.<br>Example:<br>`<element>`<br>`  <name>`<br>`  dds.transport.zrtps.compression_library`<br>`  </name>`<br>`  <value>AUTOMATIC_COMPRESSION</value>`<br>`</element>` |

## Table 6.1 Configuration Properties for ZRTPS Plugin

| Property Name | Property Value and Description |
|---|---|
| dds.transport.zrtps.<br>compression_library.small_packets<br>dds.transport.zrtps.<br>compression_library.medium_packets<br>dds.transport.zrtps.<br>compression_library.large_packets | Specifies the compression algorithm to be used for small/medium/large RTPS packages. These compression algorithms are in external libraries. The ZRTPS transport only uses the libraries whose algorithms will be used. Required libraries must be in the path during run time so the ZRTPS transport can find them.<br>See dds.transport.zrtps.compression_library.<br>Example:<br>`<element>`<br>`<name>`<br>`dds.transport.zrtps.compression_library.small_packets`<br>`</name>`<br>`<value>ZLIB_COMPRESSION</value>`<br>`</element>`<br>`<element>`<br>`<name>`<br>`dds.transport.zrtps.compression_library.medium_packets`<br>`</name>`<br>`<value>ZLIB_COMPRESSION</value>`<br>`</element>`<br>`<element>`<br>`<name>`<br>`dds.transport.zrtps.compression_library.large_packets`<br>`</name>`<br>`<value>ZLIB_COMPRESSION</value>`<br>`</element>` |
| dds.transport.zrtps.external_library | Sets the name of a user's external library that is to be located and used by the ZRTPS transport plugin. See 6.2.2 Configuring the External Compression Library on page 68.<br>Example:<br>`<element>`<br>`<name>dds.transport.zrtps.external_library</name>`<br>`<value>external_library.dll</value>`<br>`</element>` |
| dds.transport.zrtps.low_mark,<br>dds.transport.zrtps.high_mark | Specifies the size for small, medium, and large RTPS packages. RTPS packages whose size is <= **low_mark** are considered small packages. Package sizes > **low_mark** and <= **high_mark** are considered medium packages. Package sizes > **high_mark** are considered large packages.<br>The value for each property is an integer representing a number of bytes.<br>Example:<br>`<element>`<br>`<name>dds.transport.zrtps.low_mark</name>`<br>`<value>128</value>`<br>`</element>`<br>`<element>`<br>`<name>dds.transport.zrtps.high_mark</name>`<br>`<value>512</value>`<br>`</element>` |

**Table 6.1 Configuration Properties for ZRTPS Plugin**

| Property Name | Property Value and Description |
|---|---|
| dds.transport.zrtps.verbosity | The verbosity for the plugin, for debugging purposes.<br>• -1: Silent<br>• 0: Exceptions only (default)<br>• 1: Warnings<br>• 2 and up: Debug<br>**Example:**<br>`<element>`<br>`<name>dds.transport.zrtps.verbosity>/name>`<br>`<value>0</value>`<br>`</element>`<br>Note: the ZRTPS logging verbosity is per application. The last *DomainParticipant* using ZRTPS and **explicitly setting this property** will apply that setting to all the *DomainParticipants* using ZRTPS within the application. If not explicitly set, the verbosity will be left unchanged. Therefore, if no *DomainParticipant* has configured the ZRTPS verbosity, it will be left to the default value. |
| dds.transport.zrtps.property_validation_action | Optional.<br>By default, property names given in the PropertyQoSPolicy are validated to avoid using incorrect or unknown names (for example, due to a typo). This property configures the validation of the property names associated with the plugin:<br>• VALIDATION_ACTION_EXCEPTION: validate the properties. Upon failure, log errors and fail.<br>• VALIDATION_ACTION_SKIP: skip validation.<br>• VALIDATION_ACTION_WARNING: validate the properties. Upon failure, log warnings and do not fail.<br>If this property is not set, the plugin property validation behavior will be the same as that of the *DomainParticipant*, which by default is VALIDATION_ACTION_EXCEPTION. See the "Property Validation" section in the RTI Connext Core Libraries User's Manual. |

## 6.2.1 Configuring the ZRTPS Transport Plugin's 'Subtransport' Property

The required property, **dds.transport.zrtps.subtransport**, specifies the plugin to be loaded by the ZRTPS transport. Supported subtransports are **UDPv4** or a user-specified transport, as described in the following sections. *Connext* builtin transports other than UDPv4 are not currently supported as subtransports.

> The LBRTPS Transport Plugin cannot be used as a subtransport to ZRTPS. If you want to use both plugins, see 5.2.1.2 Using ZRTPS as a Subtransport on page 57.

### 6.2.1.1 Using UDPv4 as a Subtransport

To load the *Connext* UDPv4 built-in transport, use the value **UDPv4**. If you want the UDPv4 transport to be created with multicast support, also set **dds.transport.zrtps.UDPv4.multicast_enabled** to 1.

For example:

```
<element>
    <name>dds.transport.zrtps.subtransport</name>
    <value>UDPv4</value>
```

```
</element>
<element>
    <name>dds.transport.zrtps.UDPv4.multicast_enabled</name>
    <value>1</value>
</element>
```

#### 6.2.1.2 Using a User-Specified Subtransport

To load a user-provided transport plugin, provide a value for **dds.transport.zrtps.subtransport**, then use that same value as a prefix to define these two additional properties:

- **dds.transport.zrtps.<*prefix*>.library**
- **dds.transport.zrtps.<*prefix*>.create_function**

For example, to specify the value, **testplugin**:

```
<element>
    <name>dds.transport.zrtps.subtransport</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.zrtps.testplugin.library</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.zrtps.testplugin.create_function</name>
    <value>TestPlugin_Transport_create_plugin</value>
</element>
```

### 6.2.2 Configuring the External Compression Library

The ZRTPS transport plugin loads the external compression library defined in the QoS profile (see dds.transport.zrtps.compression_library). The transport will try to detect the following three functions, which must be implemented in the external library:


- 6.2.2.1 ZRTPS_Transport_external_calculate_length below
- 6.2.2.2 ZRTPS_Transport_external_compress on the next page
- 6.2.2.3 ZRTPS_Transport_external_uncompress on the next page


#### 6.2.2.1 ZRTPS_Transport_external_calculate_length

This function is called before compressing the data.

```
int ZRTPS_Transport_external_calculate_length(int data_length);
```

Parameters:

- **data_length** is the size of the data that will be compressed.

Returns:

- Maximum size of the buffer that the external library needs when the compression function is called.

#### 6.2.2.2 ZRTPS_Transport_external_compress

This function is called when data has to be compressed.

```
int ZRTPS_Transport_external_compress(char *dst,
                      int *dst_length,
                      const char *src,
                      int src_length,
                      int compression_level);
```

Parameters:

- **dst** and **src** are pointers to the destination and source buffers, respectively.
- **dst_length** is the size of the destination buffer; this size is returned by **ZRTPS_Transport_external_calculate_length.dst_length** must return the actual size of the compressed data.
- **compression_level** is a value between 1 and 9 (inclusive). Some compression libraries use this and others do not.

Returns:

- 0 if successful, -1 if unsuccessful.

#### 6.2.2.3 ZRTPS_Transport_external_uncompress

This function is called to uncompress data.

```
int ZRTPS_Transport_external_uncompress(char *dst,
                           int *dst_length,
                       const char *src,
                    int src_length);
```

Parameters:

- **dst** and **src** are pointers to the destination and source buffers, respectively.
- **dst_length** is the length of the output buffer. After the uncompression, this variable stores the length of the uncompressed data.
- **src_length** is the size of the compressed data in the source buffer.

Returns:

- 0 if successful, -1 if unsuccessful. If successful, **dst_length** must be set to the actual size of the uncompressed data.