

RTI Limited Bandwidth Plugins

User's Manual

Version 6.0.1



© 2020 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
March 2020.

Trademarks

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

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

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

The security features of this product include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

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 4

Chapter 3 Limited Bandwidth Participant Discovery Plugin

3.1 Creating the LBDP Plugin Configuration File	6
3.2 Configuring the LBDP Plugin in Connex DDS	9
3.3 Optimizing the Plugin	11
3.3.1 Preemptive Heartbeats and NACKs	11
3.3.2 Initial Announcements	12
3.3.3 Liveliness	13

Chapter 4 Limited Bandwidth Endpoint Discovery Plugin

4.1 Creating the LBED Plugin Configuration File	14
4.2 Configuring the LBED Plugin in Connex DDS	25
4.3 Optimizing the Plugin	28

Chapter 5 Limited Bandwidth RTPS Transport Plugin

5.1 Understanding the RTPS Message Header	29
5.1.1 Submessage Structure	31
5.2 Configuring the LBRTPS Transport	32
5.2.1 Configuring the LBRTPS Transport Plugin's 'Subtransport' Property	36
5.2.1.1 Using UDPv4 as a Subtransport	37
5.2.1.2 Using ZRTPS as a Subtransport	37
5.2.1.3 Using a User-Specified Subtransport	39

5.2.2 Configuring the LBRTPS Transport Plugin's 'reduce_guidPrefix' Property	39
Chapter 6 Compression Real-Time Publish Subscribe Transport Plugin	
6.1 Transport-Related Limitations	43
6.2 Configuring the ZRTPS Transport	43
6.2.1 Configuring the ZRTPS Transport Plugin's 'Subtransport' Property	48
6.2.1.1 Using UDPv4 as a Subtransport	48
6.2.1.2 Using a User-Specified Subtransport	48
6.2.2 Configuring the External Compression Library	49
6.2.2.1 ZRTPS_Transport_external_calculate_length	49
6.2.2.2 ZRTPS_Transport_external_compress	49
6.2.2.3 ZRTPS_Transport_external_uncompress	50

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 Connex*® *DDS* 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 *Connex DDS*. 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 under `<path to examples>/connext_dds/c/limited_bandwidth_plugins`:

- dil-stacking
- lbediscovery
- lbpdiscovery
- lbrtps
- zrtps

1.2 What is a Transport Plugin?

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

Connext DDS comes with standard UDPv4/IP and UDPv6/IP pluggable transports, as well as a shared memory transport; these transports are enabled by default. *Connext DDS* 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 DDS* 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 DDS* 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 participant DATA submessages.

During the Simple Endpoint Discovery phase, *Connext DDS* matches *DataWriters* and *DataReaders*. Information about each application's *DataReaders* and *DataWriters* is exchanged by sending publication/subscription declarations in DATA submessages, 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.

$$\text{Limited Bandwidth Discovery} = \text{Limited Bandwidth Participant Discovery} \\ + \text{Limited Bandwidth Endpoint Discovery}$$

1.5 Configuring Transports with the Property QoS Policy in XML

Connex DDS 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 *Connex DDS* that are not exposed through formal QoS policies. *Connex DDS* 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">
    <participant_qos>
      ...
      <property>
        <value>
          <element>
            <name>Property1</name>
            <value>example</value>
          </element>
          <element>
            <name>Property2</name>
            <value>example</value>
          </element>
          ...
        </value>
      </property>
      ...
    </participant_qos>
  </qos_profile>
</qos_library>
```

For more general information, see [Configuring QoS with XML, in the RTI Connex DDS 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® Connex® DDS*. The default installation paths are:

- macOS® systems:
/Applications/rti_connex_dds-6.0.1
- Linux systems, non-*root* user:
/home/<your user name>/rti_connex_dds-6.0.1
- Linux systems, *root* user:
/opt/rti_connex_dds-6.0.1
- Windows® systems, user without Administrator privileges:
<your home directory>\rti_connex_dds-6.0.1
- Windows systems, user with Administrator privileges:
C:\Program Files\rti_connex_dds-6.0.1

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\rtdi_connex_dds-6.0.1\bin\rtiddsgen"
```

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

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

- **<path to examples>**

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

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

Default path to the examples:

- macOS systems: **/Users/<your user name>/rti_workspace/6.0.1/examples**
- Linux systems: **/home/<your user name>/rti_workspace/6.0.1/examples**
- Windows systems: **<your Windows documents folder>\rti_workspace\6.0.1\examples**

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

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

Chapter 3 Limited Bandwidth Participant Discovery Plugin

Limited Bandwidth Participant Discovery (LBDP) 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 LBDP to work, all the participants must be known ahead of time and described in an XML file.

The LBDP 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 LBDP Plugin](#) for additional details).

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

You will need two XML files, one for the discovery plugin (see [3.1 Creating the LBDP Plugin Configuration File](#) below) and one for *Connex DDS* ([3.2 Configuring the LBDP Plugin in Connex DDS](#) on page 9).

You must link with the *dynamic* version of the *Connex DDS* libraries. See the [RTI Connex DDS Core Libraries Platform Notes](#) for details.

3.1 Creating the LBDP Plugin Configuration File

To use LBDP, 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 LBDP Plugin in Connex DDS on page 9](#); see `dds.discovery.participant.<string>.config_file`

The main structure of this file is:

```
<LBPDDiscoveryPluginProfile>
  <participant name="Participant1">
    ...
  </participant>
  <participant name="Participant2">
    ...
  </participant>
</LBPDDiscoveryPluginProfile>
```

Let's look at an example of this file (you can find it in `<path to examples>/connex_dds/c/limited_bandwidth_plugins/lbpdiscovery/LBPDDiscoveryPluginExamplePublisher.xml`).

```
<LBPDDiscoveryPluginProfile>
  <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>
</LBPDDiscoveryPluginProfile>
```

(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 *Connex DDS* documentation. For example, “See documentation on the Entity QoS policy” means you should see that section in the [RTI Connex DDS Core Libraries User's Manual](#) or the *Connex DDS* API Reference HTML documentation.

Table 3.1 Configuration Options for LBPDP Plugin

Option Name	Option Value and Description
key	<p>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.</p> <p>Example:</p> <pre data-bbox="407 478 954 583"><key> <rtps_host_id>123</rtps_host_id> <rtps_app_id>255</rtps_app_id> <rtps_instance_id>348</rtps_instance_id> </key></pre>
liveliness_lease_duration	<p>The liveliness lease duration for the participant.</p> <p>Schema:</p> <pre data-bbox="407 678 976 863"><liveliness_lease_duration> <sec>[number DURATION_ZERO_SEC DURATION_INFINITE_SEC] </sec> <nanosec>[number DURATION_ZERO_NSEC DURATION_INFINITE_NSEC] </nanosec> </liveliness_lease_duration></pre> <p>Example:</p> <pre data-bbox="407 898 919 982"><liveliness_lease_duration> <sec>100</sec> <nanosec>DURATION_ZERO_NSEC</nanosec> </liveliness_lease_duration></pre>
rtps_protocol_version	<p>The version number of the RTPS protocol being used. See documentation on the ParticipantBuiltinTopicData structure.</p> <p>Example:</p> <pre data-bbox="407 1108 711 1192"><rtps_protocol_version> <major>2</major> <minor>1</minor> </rtps_protocol_version></pre>
rtps_vendor_id	<p>The identifier of the RTPS vendor. See documentation on ParticipantBuiltinTopicData. RTI's identifier is 1, 1.</p> <p>Example:</p> <pre data-bbox="407 1287 1097 1308"><rtps_vendor_id><vendorId>1,1</vendorId></rtps_vendor_id></pre>
participant_name	<p>The name of the remote participant, as set in EntityName QoS policy.</p> <p>Example:</p> <pre data-bbox="407 1402 846 1423"><participant name="Participant1"/></pre> <p>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:</p> <pre data-bbox="407 1528 943 1577">LBPDiscoveryPluginTypeReaderListenerOnDataAvailable: Cannot find the participant in the database</pre>
product_version	<p>The version number for the plugin.</p> <p>Example (where x, y, and z represent numbers of the current release):</p> <pre data-bbox="407 1675 740 1801"><product_version> <major>6</major> <minor>x</minor> <release>y</release> <revision>z</revision> </product_version></pre>

3.2 Configuring the LBPDP Plugin in Connex DDS

This section describes how to configure the properties for the LBPDP plugin in the XML QoS Profile file used by *Connex DDS* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQosPolicy for your *Connex DDS* application's *DomainParticipant*.

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

```
<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>
...
</participant_qos>
```

[Table 3.2 LBPDP Configuration Properties for Connex DDS](#) describes the name/value pairs that you can use to configure the LBPDP plugin.

Table 3.2 LBPDP Configuration Properties for Connex DDS

Property Name	Property Value and Description
dds.discovery.participant.load_plugins	<p>Required.</p> <p>String indicating the prefix name of the plugin that will be loaded by <i>Connex DDS</i>.</p> <p>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 <code>lbpdiscovery</code>:</p> <pre data-bbox="423 491 1105 579"><element> <name>dds.discovery.participant.load_plugins</name> <value>dds.discovery.participant.lbpdiscovery</value> </element></pre>
dds.discovery.participant.<string>.library	<p>Required.</p> <p>The name of the dynamic library that contains the LBPDP plugin implementation. This library must be in the path during run time for use by <i>Connex DDS</i>.</p> <p>Set the value to rtilbpdisc.</p> <p>Example:</p> <pre data-bbox="423 779 1068 909"><element> <name> dds.discovery.participant.lbpdiscovery.library </name> <value>rtilbpdisc</value> </element></pre>
dds.discovery.participant.<string>.create_function	<p>Required.</p> <p>The name of the function that will be called by <i>Connex DDS</i> to create an instance of the LBPDP plugin.</p> <p>Set the value to DDS_LBPDiscoveryPlugin_create.</p> <p>Example:</p> <pre data-bbox="423 1083 1166 1213"><element> <name> dds.discovery.participant.lbpdiscovery.create_function </name> <value>DDS_LBPDiscoveryPlugin_create</value> </element></pre>
dds.discovery.participant.<string>.config_file	<p>Required.</p> <p>The name of the discovery configuration file, described in 3.1 Creating the LBPDP Plugin Configuration File on page 6.</p> <p>Set the value to the name of your own file.</p> <p>Example:</p> <pre data-bbox="423 1381 1117 1512"><element> <name> dds.discovery.participant.lbpdiscovery.config_file </name> <value>LBPDiscoveryPluginExampleSubscriber.xml</value> </element></pre>

Table 3.2 LBPD Configuration Properties for Connex DDS

Property Name	Property Value and Description
dds.discovery.participant. <string>.verbosity	Optional. The verbosity for the plugin, for debugging purposes. <ul style="list-style-type: none"> • -1: Silent • 0: Exceptions only (default) • 1: Warnings • 2 and up: Debug Example: <pre> <element> <name> dds.discovery.participant.lbpdiscovery.verbosity </name> <value>0</value> </element> </pre>

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 *Connex DDS* 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 *Connex DDS* 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 Preemptive Heartbeats and NACKs below](#)
- [3.3.2 Initial Announcements on the next page](#)
- [3.3.3 Liveliness on page 13](#)

3.3.1 Preemptive Heartbeats and NACKs

Each participant manages one channel with each of the other participants; the channel is used to keep the participant's liveliness. This channel is reliable and uses heartbeats and NACKs. You can reduce network bandwidth by disabling preemptive heartbeats and NACKs in this channel.

For example:

```

<participant_qos>
  <property>
    <value>
      <element>
        <name>
          dds.participant.inter_participant_data_reader.
            protocol.disable_preemptive_nack
        </name>
        <value>1</value>
      </element>
      <element>
        <name>
          dds.participant.inter_participant_data_writer.
            protocol.disable_preemptive_heartbeat
        </name>
        <value>1</value>
      </element>
    </value>
  </property>
</participant_qos>

```

3.3.2 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:

```

<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>
</participant_qos>

```

3.3.3 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:

```
<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>
</participant_qos>
```

Chapter 4 Limited Bandwidth Endpoint Discovery Plugin

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 reduce network traffic. However, for LBED to work, all the endpoints must be known ahead of time and described in an XML file.

This chapter describes how to configure the LBED Plugin and set up your *Connex DDS* application to use it.

You will need two XML files, one for the discovery plugin (see [4.1 Creating the LBED Plugin Configuration File below](#)) and one for *Connex DDS* (see [4.2 Configuring the LBED Plugin in Connex DDS on page 25](#)).

You must link with the *dynamic* version of the *Connex DDS* libraries. See the [RTI Connex DDS Core Libraries Platform Notes](#) for details.

4.1 Creating the LBED Plugin Configuration File

To use LBED, you need an XML file that describes all the remote participants and their endpoints. These remote endpoints 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 [4.2 Configuring the LBED Plugin in Connex DDS on page 25](#); see `dds.discovery.endpoint.<string>.config_file`.

The main structure of the XML file is:

```
<LBEDiscoveryPluginProfile>
  <participant name="myParticipant">
    <datareader>
      ...
    </datareader>
    ...
  </participant>
  ...
</LBEDiscoveryPluginProfile>
```

```

    <datawriter>
      ...
    </datawriter>
  </participant>
</LBEDiscoveryPluginProfile>

```

The supported configuration options for *DataReaders* and *DataWriters* are described in [Table 4.1 Configuration Options for LBED Plugin](#). The descriptions for many of these options also point out related *Connext DDS* documentation. For example, “See documentation on the Deadline QoS policy” means you should see that section in the [RTI Connext DDS Core Libraries User's Manual](#) or API Reference HTML documentation. These options can be set for *DataReaders* and *DataWriters*, unless otherwise noted in [Table 4.1 Configuration Options for LBED Plugin](#).

The following example shows an example LBED plugin configuration file, which you can find here:

**<path to examples>/connext_dds/c/
limited_bandwidth_plugins/lbediscovery/LBEDiscoveryPluginExamplePublisher.xml:**

```

<?xml version="1.0"?>
<LBEDiscoveryPluginProfile xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance">
  <participant name="myParticipant2">
    <datawriter>
      <rtps_object_id>200</rtps_object_id>
      <topic_name>Hello LBEDiscovery</topic_name>
      <type_name>DDS::String</type_name>
      <topic_keyed>>false</topic_keyed>
      <durability>
        <kind>VOLATILE_DURABILITY_QOS</kind>
        <direct_communication>>true</direct_communication>
      </durability>
      <destination_order>
        <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
        <source_timestamp_tolerance>
          <sec>30</sec>
          <nanosec>0</nanosec>
        </source_timestamp_tolerance>
      </destination_order>
      <presentation>
        <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
        <coherent_access>>false</coherent_access>
        <ordered_access>>false</ordered_access>
      </presentation>
      <deadline>
        <period>
          <sec>DURATION_INFINITE_SEC</sec>
          <nanosec>DURATION_INFINITE_NSEC</nanosec>
        </period>
      </deadline>
      <latency_budget>
        <duration>
          <sec>0</sec>
          <nanosec>0</nanosec>

```

```

        </duration>
    </latency_budget>
    <liveliness>
        <kind>AUTOMATIC_LIVELINESS_QOS</kind>
        <lease_duration>
            <sec>DURATION_INFINITE_SEC</sec>
            <nanosec>DURATION_INFINITE_NSEC</nanosec>
        </lease_duration>
    </liveliness>
    <reliability>
        <kind>RELIABLE_RELIABILITY_QOS</kind>
        <max_blocking_time>
            <sec>0</sec>
            <nanosec>100000000</nanosec>
        </max_blocking_time>
        <acknowledgment_kind>
            PROTOCOL_ACKNOWLEDGMENT_MODE
        </acknowledgment_kind>
    </reliability>
    <ownership>
        <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
    </ownership>
    <ownership_strength>
        <value>0</value>
    </ownership_strength>
    <lifespan>
        <duration>
            <sec>DURATION_INFINITE_SEC</sec>
            <nanosec>DURATION_INFINITE_NSEC</nanosec>
        </duration>
    </lifespan>
    <durability_service>
        <service_cleanup_delay>
            <sec>0</sec>
            <nanosec>0</nanosec>
        </service_cleanup_delay>
        <history_kind>KEEP_LAST_HISTORY_QOS</history_kind>
        <history_depth>1</history_depth>
        <max_samples>LENGTH_UNLIMITED</max_samples>
        <max_instances>LENGTH_UNLIMITED</max_instances>
        <max_samples_per_instance>
            LENGTH_UNLIMITED
        </max_samples_per_instance>
    </durability_service>
    <service>NO_SERVICE</service>
</datawriter>
</participant>
</LBEDiscoveryPluginProfile>

```

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
deadline	<p>Maximum time between data samples. See documentation on the Deadline QoS policy.</p> <p>Schema:</p> <pre data-bbox="321 457 737 590"><deadline> <period> <sec>[number]</sec> <nanosec>[number]</nanosec> </period> </deadline></pre> <p>Example:</p> <pre data-bbox="321 625 902 758"><deadline> <period> <sec>DURATION_INFINITE_SEC</sec> <nanosec>DURATION_INFINITE_NSEC</nanosec> </period> </deadline></pre>
destination_order	<p>Controls how <i>Connex DDS</i> will deal with data sent by multiple <i>DataWriters</i> for the same topic. See documentation on the DestinationOrder QoS policy.</p> <p>Schema:</p> <pre data-bbox="321 877 984 1073"><destination_order> <kind>[BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS] </kind> <source_timestamp_tolerance> <sec>[number]</sec> <nanosec>[number]</nanosec> </source_timestamp_tolerance> </destination_order></pre> <p>Example:</p> <pre data-bbox="321 1108 1032 1262"><destination_order> <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind> <source_timestamp_tolerance> <sec>30</sec> <nanosec>0</nanosec> </source_timestamp_tolerance> </destination_order></pre>
disable_positive_acks	<p>Whether or not to disable positive ACKs. See documentation on the DataWriterProtocol and DataReaderProtocol QoS policies.</p> <p>Schema:</p> <pre data-bbox="321 1360 651 1472"><protocol> <disable_positive_acks> [true false] </disable_positive_acks> </protocol></pre> <p>Example:</p> <pre data-bbox="321 1507 651 1619"><protocol> <disable_positive_acks> true </disable_positive_acks> </protocol></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
durability	<p>Specifies if <i>Connex DDS</i> will store and deliver previously published data to new/late-joining <i>DataReaders</i>. See documentation on the Durability QoS policy.</p> <p>Schema:</p> <pre><durability> <kind> [VOLATILE_DURABILITY_QOS TRANSIENT_LOCAL_DURABILITY_QOS TRANSIENT_DURABILITY_QOS PERSISTENT_DURABILITY_QOS] </kind> <direct_communication> [true false] </direct_communication> </durability></pre> <p>Example:</p> <pre><durability> <kind>VOLATILE_DURABILITY_QOS</kind> <direct_communication>true</direct_communication> </durability></pre>
durability_ser- vice	<p>Various settings to configure the external <i>Persistence Service</i> used by <i>Connex DDS</i> for <i>DataWriters</i> with a Durability setting of persistent. See documentation on the Durability QoS policy.</p> <p>Schema:</p> <pre><durability_service> <service_cleanup_delay> <sec>[number]</sec> <nanosec>[number]</nanosec> </service_cleanup_delay> <history_kind> [KEEP_LAST_HISTORY_QOS KEEP_ALL_HISTORY_QOS] </history_kind> <history_depth>[number]</history_depth> <max_samples>[number]</max_samples> <max_instances>[number]</max_instances> <max_samples_per_instance> [number] </max_samples_per_instance> </durability_service></pre> <p>Example:</p> <pre><durability_service> <service_cleanup_delay> <sec>0</sec> <nanosec>0</nanosec> </service_cleanup_delay> <history_kind>KEEP_LAST_HISTORY_QOS</history_kind> <history_depth>1</history_depth> <max_samples>LENGTH_UNLIMITED</max_samples> <max_instances>LENGTH_UNLIMITED</max_instances> <max_samples_per_instance> LENGTH_UNLIMITED </max_samples_per_instance> </durability_service></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
group_data	<p>Attaches arbitrary application data (a buffer of bytes) to discovery meta-data. See documentation on the GroupData QoS policy.</p> <p>Schema:</p> <pre><group_data mode=["hexadecimal" "string"]> <value>[information]</value> </group_data></pre> <p>Example:</p> <pre><group_data mode="string"> <value>Dump information</value> </group_data></pre>
latency_budget	<p>Suggests to the middleware how much time is allowed to deliver data. See documentation on the LatencyBudget QoS policy.</p> <p>Schema:</p> <pre><latency_budget> <duration> <sec>[number]</sec> <nanosec>[number]</nanosec> </duration> </latency_budget></pre> <p>Example:</p> <pre><latency_budget> <duration> <sec>0</sec> <nanosec>0</nanosec> </duration> </latency_budget></pre>
lifespan	<p>Specifies how long <i>Connex DDS</i> should consider data sent by an user application to be valid. See documentation on the Lifespan QoS policy.</p> <p>Applies only to <i>DataWriters</i>.</p> <p>Schema:</p> <pre><lifespan> <duration> <sec>[number]</sec> <nanosec>[number]</nanosec> </duration> </lifespan></pre> <p>Example:</p> <pre><lifespan> <duration> <sec>DURATION_INFINITE_SEC</sec> <nanosec> DURATION_INFINITE_NSEC</nanosec> </duration> </lifespan></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
liveliness	<p>Liveliness specifies how <i>Connex DDS</i> determines whether a <i>DataWriter</i> is “alive.” See documentation on the Liveliness QoS policy.</p> <p>Schema:</p> <pre><liveliness> <kind>[AUTOMATIC_LIVELINESS_QOS MANUAL_BY_PARTICIPANT_LIVELINESS_QOS MANUAL_BY_TOPIC_LIVELINESS_QOS] </kind> <lease_duration> <sec>[number]</sec> <nanosec>[number]</nanosec> </lease_duration> </liveliness></pre> <p>Example:</p> <pre><liveliness> <kind>AUTOMATIC_LIVELINESS_QOS</kind> <lease_duration> <sec>DURATION_INFINITE_SEC</sec> <nanosec> DURATION_INFINITE_NSEC</nanosec> </lease_duration> </liveliness></pre>
multicast	<p>List of multicast addresses on which the <i>DataReader</i> will listen for data. See documentation on the TransportMulticast QoS policy.</p> <p>Applies only to <i>DataReaders</i>.</p> <p>Schema:</p> <pre><multicast> <number_of_elements>[number]</number_of_elements> <locator> <kind>[UDPv4]</kind> <address>[IP address]</address> <port>[IP port]</port> </locator> <locator> ... </locator> ... </multicast></pre> <p>Example:</p> <pre><multicast> <number_of_elements>2</number_of_elements> <locator> <kind>UDPv4</kind> <address>192.168.1.0</address> <port>7400</port> </locator> <locator> <kind>UDPv4</kind> <address>192.168.200.0</address> <port>7401</port> </locator> </multicast></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
ownership, ownership_strength	<p>Specifies if <i>DataReaders</i> for a topic can receive data from multiple <i>DataWriters</i> at the same time. See documentation on the Ownership and Ownership Strength QoS policies.</p> <p>Applies to <i>DataReaders</i> (except Strength option) and <i>DataWriters</i>.</p> <p>Schema:</p> <pre><ownership> <kind> [SHARED_OWNERSHIP_QOS EXCLUSIVE_OWNERSHIP_QOS] </kind> </ownership> <ownership_strength> <value>[number]</value> </ownership_strength></pre> <p>Example:</p> <pre><ownership> <kind>EXCLUSIVE_OWNERSHIP_QOS</kind> </ownership> <ownership_strength> <value>0</value> </ownership_strength></pre>
partitions	<p>Stores a set of partition names that identify the partitions of which the entity is a member. See documentation on the Partition QoS policy.</p> <p>Schema:</p> <pre><partitions> <number_of_elements>[number]</number_of_elements> <partition> <name>[string]</name> </partition> <partition> ... </partition> ... </partitions></pre> <p>Example:</p> <pre><partitions> <number_of_elements>2</number_of_elements> <partition> <name>Partition1</name> </partition> <partition> <name>Partition2</name> </partition> </partitions></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
presentation	<p>Controls how <i>Connex DDS</i> presents data received by an application to the <i>DataReaders</i> of the data. See documentation on the Presentation QoS policy.</p> <p>Schema:</p> <pre><presentation> <access_scope>[INSTANCE_PRESENTATION_QOS TOPIC_PRESENTATION_QOS GROUP_PRESENTATION_QOS] </access_scope> <coherent_access>[true false]</coherent_access> <ordered_access>[true false]</ordered_access> </presentation></pre> <p>Example:</p> <pre><presentation> <access_scope>INSTANCE_PRESENTATION_QOS</access_scope> <coherent_access>>false</coherent_access> <ordered_access>>false</ordered_access> </presentation></pre>
property	<p>Stores name/value (string) pairs that can be used to configure certain parameters of <i>Connex DDS</i> that are not exposed through formal QoS policies. See documentation on the Property QoS policy.</p> <p>Schema:</p> <pre><property> <number_of_elements>[number]</number_of_elements> <element> <name>[string]</name> <value>[string]</value> <propagate>[true false]</propagate> </element> <element> ... </element> ... </property></pre> <p>Example:</p> <pre><property> <number_of_elements>2</number_of_elements> <element> <name>Coin</name> <value>Euro</value> <propagate>>false</propagate> </element> <element> <name>Country</name> <value>Spain</value> <propagate>>false</propagate> </element> </property></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
reliability	<p>Determines whether or not data published by a <i>DataWriter</i> will be reliably delivered to matching <i>DataReaders</i>. See documentation on the Reliability QoS policy.</p> <p>Schema:</p> <pre data-bbox="316 489 914 806"><reliability> <kind>[RELIABLE_RELIABILITY_QOS BEST_EFFORT_RELIABILITY_QOS] </kind> <max_blocking_time> <sec>[number]</sec> <nanosec>[number]</nanosec> </max_blocking_time> <acknowledgment_kind> [PROTOCOL_ACKNOWLEDGMENT_MODE APPLICATION_AUTO_ACKNOWLEDGMENT_MODE APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE] </acknowledgment_kind> </reliability></pre> <p>Example:</p> <pre data-bbox="316 846 831 1066"><reliability> <kind> RELIABLE_RELIABILITY_QOS </kind> <max_blocking_time> <sec>0</sec> <nanosec>100000000</nanosec> </max_blocking_time> <acknowledgment_kind> PROTOCOL_ACKNOWLEDGMENT_MODE </acknowledgment_kind> </reliability></pre>
rtps_object_id	<p>Determines the RTPS object ID of a <i>DataWriter/DataReader</i>. See documentation on the DataWriterProtocol QoS policy.</p> <p>Schema:</p> <pre data-bbox="316 1157 914 1188"><rtps_object_id>[positive number]</rtps_object_id></pre> <p>Example:</p> <pre data-bbox="316 1224 724 1255"><rtps_object_id>1</rtps_object_id></pre>
rtps_protocol_version	<p>Version of the RTPS protocol used by the entity.</p> <p>Schema:</p> <pre data-bbox="316 1339 605 1409"><rtps_protocol_version> [number].[number] </rtps_protocol_version></pre> <p>Example:</p> <pre data-bbox="316 1444 914 1476"><rtps_protocol_version>0.0</rtps_protocol_version></pre>
service	<p>Specifies the use of an external persistence service.</p> <p>Schema: <service>[NO_SERVICE PERSISTENCE]</service></p> <p>Example: <service>NO_SERVICE</service></p>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
time_based_filter	<p>Set by a <i>DataReader</i> to limit the number of new data values received over a period of time. See documentation on the TimeBasedFilter QoS policy.</p> <p>Applies only to <i>DataReaders</i>.</p> <p>Schema:</p> <pre data-bbox="321 527 737 657"><time_based_filter> <minimum_separation> <sec>[number]</sec> <nanosec>[number]</nanosec> </minimum_separation> </time_based_filter></pre> <p>Example:</p> <pre data-bbox="321 695 667 825"><time_based_filter> <minimum_separation> <sec>0</sec> <nanosec> 0</nanosec> </minimum_separation> </time_based_filter></pre>
topic_data	<p>Attaches arbitrary application data (a buffer of bytes) to discovery meta-data. See documentation on the TopicData QoS policy.</p> <p>Applies only to <i>DataReaders</i>.</p> <p>Schema:</p> <pre data-bbox="321 957 821 1024"><topic_data mode=["hexadecimal" "string"]> <value>[information]</value> </topic_data></pre> <p>Example:</p> <pre data-bbox="321 1062 737 1129"><topic_data mode="string"> <value>Dump information</value> </topic_data></pre>
topic_keyed	<p>Indicates whether the data type has a key.</p> <p>Schema:</p> <pre data-bbox="321 1213 786 1245"><topic_keyed>[true false]</topic_keyed></pre> <p>Example:</p> <pre data-bbox="321 1276 699 1308"><topic_keyed>>false</topic_keyed></pre>
topic_name	<p>The name of the entity's topic. See documentation on the topic parameter in create_datawriter() or create_datareader().</p> <p>Schema:</p> <pre data-bbox="321 1398 712 1430"><topic_name>[string]</topic_name></pre> <p>Example:</p> <pre data-bbox="321 1461 773 1493"><topic_name>Example topic</topic_name></pre>
type_name	<p>The name of the entity's type. See documentation on the type_name parameter in create_topic().</p> <p>Schema:</p> <pre data-bbox="321 1577 688 1608"><type_name>[string]</type_name></pre> <p>Example:</p> <pre data-bbox="321 1640 737 1671"><type_name>Example type</type_name></pre>

Table 4.1 Configuration Options for LBED Plugin

Option Name	Option Value and Description
unicast	<p>List of all unicast addresses on which the <i>DataReader/DataWriter</i> will listen for data. See documentation on the TransportUnicast QoS policy.</p> <p>Schema:</p> <pre data-bbox="321 489 951 747"><unicast> <number_of_elements>[number]</number_of_elements> <locator> <kind>[UDPv4]</kind> <address>[IP address]</address> <port>[IP port]</port> </locator> <locator> ... </locator> ... </unicast></pre> <p>Example:</p> <pre data-bbox="321 789 867 1068"><unicast> <number_of_elements>2</number_of_elements> <locator> <kind>UDPv4</kind> <address>82.123.32.4 </address> <port>7400</port> </locator> <locator> <kind>UDPv4</kind> <address>82.123.32.7</address> <port>7401</port> </locator> </unicast></pre>
user_data	<p>Attaches arbitrary application data (a buffer of bytes) to discovery meta-data. See documentation on the UserData QoS policy.</p> <p>Schema:</p> <pre data-bbox="321 1163 808 1230"><user_data mode=["hexadecimal" "string"]> <value>[information]</value> </user_data></pre> <p>Example:</p> <pre data-bbox="321 1272 678 1331"><user_data mode="hexadecimal"> <value>04FC8D922E</value> </user_data></pre>

4.2 Configuring the LBED Plugin in Connex DDS

This section describes how to configure the properties for the LBED plugin in the XML QoS Profile file used by *Connex DDS* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQoSPolicy for your application's *DomainParticipant*.

Let's look at part of an example XML file, which you can find in `<path to examples>/connex_dds/c/limited_bandwidth_plugins/lbediscovery/USER_QOS_PROFILES.xml`:

```
<qos_profile name="LBEDiscoveryPluginExampleSubscriber_Profile">
  ...
  <participant_qos>
    <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.lbediscovery.config_file
    </name>
    <value>
        LBEDiscoveryPluginExamplePublisher.xml
    </value>
</element>
<element>
    <name>
        dds.discovery.endpoint.load_plugins
    </name>
    <value>
        dds.discovery.endpoint.lbediscovery
    </value>
</element>
</value>
</property>
</participant_qos>

```

Table 4.2 LBED Configuration Properties for Connex DDS describes the name/value pairs that you can use to configure the LBED plugin.

Table 4.2 LBED Configuration Properties for Connex DDS

Property Name	Property Value and Description
dds.discovery.endpoint.load_plugins	<p>Required.</p> <p>String indicating the prefix name of the plugin that will be loaded by <i>Connex DDS</i>.</p> <p>Set the value to <code>dds.discovery.endpoint.<string></code>, where <code><string></code> 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:</p> <pre> <element> <name>dds.discovery.endpoint.load_plugins</name> <value>dds.discovery.endpoint.lbediscovery</value> </element> </pre>

Table 4.2 LBED Configuration Properties for Connex DDS

Property Name	Property Value and Description
dds.discovery.endpoint. <string>.library	Required. 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 <i>Connex DDS</i> . Set the value to rtilbedisc . Example: <pre> <element> <name> dds.discovery.endpoint.lbediscovery.library </name> <value>rtilbedisc</value> </element> </pre>
dds.discovery.endpoint. <string>.create_function	Required. The name of the function that will be called by <i>Connex DDS</i> to create an instance of the LBED plugin. Set the value to DDS_LBEDiscoveryPlugin_create . Example: <pre> <element> <name> dds.discovery.endpoint.lbediscovery.create_function </name> <value>DDS_LBEDiscoveryPlugin_create</value> </element> </pre>
dds.discovery.endpoint. <string>.config_file	Required. The name of the discovery configuration file, described in 4.1 Creating the LBED Plugin Configuration File on page 14 . Set the value to the name of your own file. Example: <pre> <element> <name> dds.discovery.endpoint.lbediscovery.config_file </name> <value>LBEDiscoveryPluginExampleSubscriber.xml</value> </element> </pre>
dds.discovery.endpoint. <string>.verbosity	The verbosity for the plugin, for debugging purposes. <ul style="list-style-type: none"> • -1: Silent • 0: Exceptions only (default) • 1: Warnings • 2 and up: Debug Example: <pre> <element> <name> dds.discovery.endpoint.lbediscovery.verbosity </name> <value>0</value> </element> </pre>

In addition to setting the properties described above, the **builtin_discovery_plugins** mask (set in the DiscoveryConfigQosPolicy) should be set to SPDP. 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 LBED plugin replaces the need for the SEDP, so the

builtin_discovery_plugins should be set to SPDP. This tells *Connex DDS* to only use the SPDP for participant discovery, since endpoint discovery will use the LBED plugin. If you are using both the LBED plugin and the LBPD plugin, this mask should be set to MASK_NONE.

4.3 Optimizing the Plugin

You can reduce network bandwidth by changing some *Connex DDS* properties in the XML file, **USER_QOS_PROFILE.xml**. You can see an example of this user profile in `<path to examples>/connex_dds/c/limited_bandwidth_plugins/lbediscovery/USER_QOS_PROFILES.xml`.

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. *Connex DDS* 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 [RTPS Wire Protocol Specification v2.1](#).

You must link with the *dynamic* version of the *Connex DDS* libraries. See the [RTI Connex DDS Core Libraries Platform Notes](#) for details.

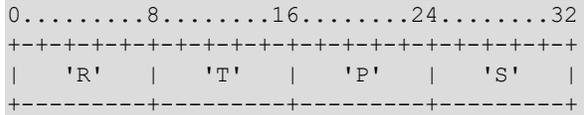
5.1 Understanding the RTPS Message Header

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



- **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

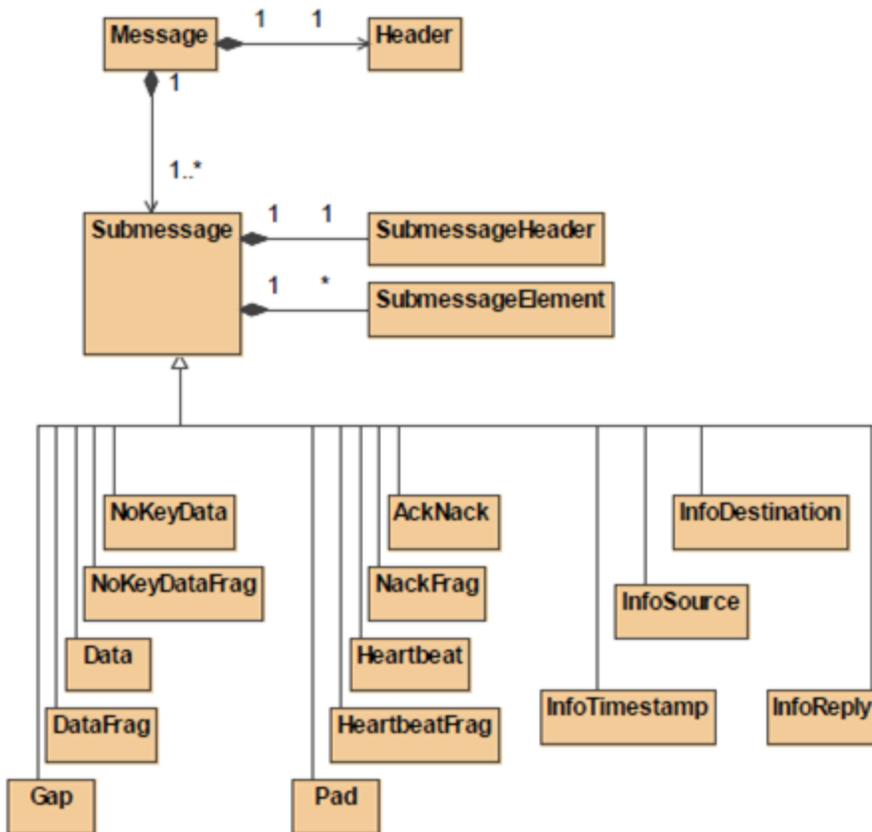
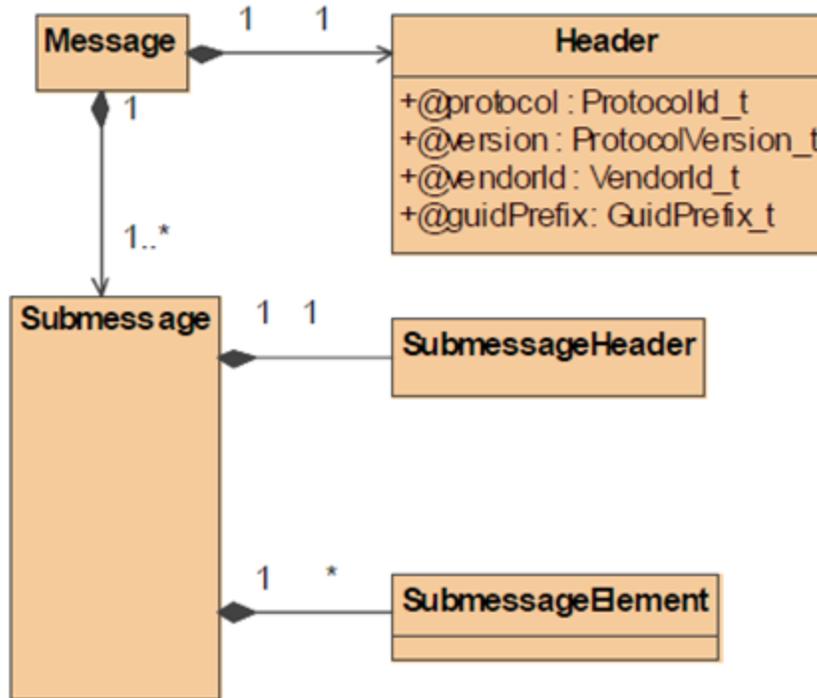


Figure 5.2: RTPS Message Header Structure



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 *Connex DDS* Transport API; for more information, please see [Transport Plugins, in the RTI Connex DDS Core Libraries User's Manual](#).

Before we describe the name/value pairs that can be used in the Property QoS policy 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">
    <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>
    ...
</participant_qos>
</qos_profile>
</qos_library>

```

When using the above example, *Connex DDS* 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 *Connex DDS* UDPv4 transport. The LBRTPS transport is registered in *Connex DDS* with the participant that uses this QoS profile.

Connex DDS does not assign initial peers to the LBRTPS transport plugin. You can set the initial peers with `NDDS_DISCOVERY_PEERS`, as described in [Discovery, in the RTI Connex DDS 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.load_plugins	<p>Required.</p> <p>Comma-separated strings indicating the prefix names of all plugins that will be loaded by <i>Connex DDS</i>.</p> <p>Must be set to dds.transport.lbrtps.</p> <p>Example:</p> <pre data-bbox="370 499 889 590"><element> <name>dds.transport.load_plugins</name> <value>dds.transport.lbrtps</value> </element></pre>
dds.transport.lbrtps.subtransport	<p>Required.</p> <p>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 36.</p>
dds.transport.lbrtps.library	<p>Required.</p> <p>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 <i>Connex DDS</i>.</p> <p>Example:</p> <pre data-bbox="370 898 911 982"><element> <name>dds.transport.lbrtps.library</name> <value>rtilbrtps</value> </element></pre>
dds.transport.lbrtps.create_function	<p>Required.</p> <p>The name of the function that will be called by <i>Connex DDS</i> to create an instance of the LBRTPS transport. The function must have the prototype of <code>NDDS_Transport_create_plugin</code>.</p> <p>Must be set to <code>LBRTPS_Transport_create_plugin</code>.</p> <p>Example:</p> <pre data-bbox="370 1182 1008 1266"><element> <name>dds.transport.lbrtps.create_function</name> <value>LBRTPS_Transport_create_plugin</value> </element></pre>
dds.transport.lbrtps.aliases	<p>Required.</p> <p>Aliases used to register the LBRTPS transport plugin with the <i>DomainParticipant</i>. The transport must have been created by the <code>dds.transport.lbrtps.create_function</code>. Aliases should be specified as comma-separated strings, with each comma delimiting an alias.</p> <p>An example alias for the LBRTPS transport, working over a <i>Connex DDS</i> UDPv4 transport: <code>lbrtps.udpv4</code>.</p> <p>Example:</p> <pre data-bbox="370 1497 911 1581"><element> <name>dds.transport.lbrtps.aliases</name> <value>lbrtps.udpv4</value> </element></pre>
The following properties are optional and appear in alphabetical order.	

Table 5.1 Configuration Properties for LBRTPS Plugin

Property Name	Property Value and Description
dds.transport.lbrtps.rtps_header.eliminate_protocol	<p>Whether or not to eliminate the 4-byte protocol field in the RTPS header.</p> <p>Must be a boolean value: true or false.</p> <p>Example:</p> <pre> <element> <name> dds.transport.lbrtps.rtps_header.eliminate_protocol </name> <value>true</value> </element> </pre>
dds.transport.lbrtps.rtps_header.eliminate_vendorId	<p>Whether or not to eliminate the 2-byte vendorId field in the RTPS header.</p> <p>Must be a boolean value: true or false.</p> <p>Example:</p> <pre> <element> <name> dds.transport.lbrtps.rtps_header.eliminate_vendorId </name> <value>true</value> </element> </pre>
dds.transport.lbrtps.rtps_header.eliminate_version	<p>Whether or not to eliminate the 2-byte version field in the RTPS header.</p> <p>Must be a boolean value: true or false.</p> <p>Example:</p> <pre> <element> <name> dds.transport.lbrtps.rtps_header.eliminate_version </name> <value>true</value> </element> </pre>
dds.transport.lbrtps.rtps_header.reduce_guidPrefix	<p>The reduce_guidPrefix field is comprised of 12 bytes that represent 3 different fields of 4 bytes each: hostId, appId and instanceId.</p> <p>See 5.2.1 Configuring the LBRTPS Transport Plugin's 'Subtransport' Property on the next page</p>
dds.transport.lbrtps.submessage_header.combine_submessageId_with_flags	<p>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.</p> <p>Must be set to a boolean value: true or false.</p> <p>Example:</p> <pre> <element> <name> dds.transport.lbrtps.submessage_header.combine_submessageId_with_flags </name> <value>true</value> </element> </pre>

Table 5.1 Configuration Properties for LBRTPS Plugin

Property Name	Property Value and Description
<p>dds.transport.lbrtps.submessages.reduce_entitiesId</p>	<p>Many submessage kinds include two fields: readerId and writerId.</p> <p>In the RTPS protocol specification, these fields are mapped to structure:</p> <pre>struct { octet[3] entityKey; octet entityKind; } EntityId_t;</pre> <p>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 <i>DataReaders</i> and <i>DataWriters</i> you will have.</p> <p>If you will have no more than 2048 <i>DataReaders</i> and 2048 <i>DataWriters</i>, you can reduce the size of each of these fields from four to two bytes. And if you will have no more than 8 <i>DataReaders</i> and 8 <i>DataWriters</i>, you can reduce each field to only one byte. The math involved is explained below.</p> <p>Five bits are always needed for the entityKind. If you have no more than 2048 <i>DataReaders</i> and 2048 <i>DataWriters</i>, 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.</p> <p>Suppose you have no more than 8 <i>DataReaders</i> and 8 <i>DataWriters</i>. 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.</p> <p>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.</p> <p>For example, to reduce both readerId and writerId to 12 bits each:</p> <pre><element> <name>dds.transport.lbrtps.submessages.reduce_entitiesId</name> <value>12,12</value> </element></pre>
<p>dds.transport.lbrtps.submessages.reduce_sequenceNumber</p>	<p>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.</p> <p>The value must be an integer between 16 and 64 that specifies the desired size of the sequence number, in bits.</p> <p>Example:</p> <pre><element> <name> dds.transport.lbrtps.submessages.reduce_sequenceNumber </name> <value>32</value> </element></pre>
<p>dds.transport.lbrtps.verbosity</p>	<p>The verbosity for the plugin, for debugging purposes.</p> <ul style="list-style-type: none"> • -1: Silent • 0: Exceptions only (default) • 1: Warnings • 2 and up: Debug <p>Example:</p> <pre><element> <name>dds.transport.lbrtps.verbosity</name> <value>0</value> </element></pre>

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 *Connex DDS* 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 42](#), 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. *Connex DDS* uses RTPS packages to send data over the network.

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

You can configure the Z RTPS 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 4](#) and `<architecture>` is one of the supported architecture strings listed in the *RTI Connex DDS 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 Z RTPS 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 *Connex DDS* UDPv4 transport, or a custom transport created using the *Connex DDS* Transports API. You can configure *Connex DDS* 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 *Connex DDS* core libraries. See the *RTI Connex DDS 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 37](#).
- 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 *Connex DDS* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQoSPolicy for your application's *DomainParticipant*.

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">
    <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>
    ...
</participant_qos>
</qos_profile>
</qos_library>

```

When using the above QoS profile, *Connex DDS* 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 *Connex DDS* 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 *Connex DDS* with the participant that uses this QoS profile.

Connex DDS 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 Connex DDS 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	<p>Required.</p> <p>Comma-separated strings indicating the prefix names of all plugins to be loaded by <i>Connex DDS</i>.</p> <p>Set the value to dds.transport.zrtps.</p> <p>Example:</p> <pre><element> <name>dds.transport.load_plugins</name> <value>dds.transport.zrtps</value> </element></pre>
dds.transport.zrtps.library	<p>Required.</p> <p>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 <i>Connex DDS</i>.</p> <p>The value must be rtizrtps.</p> <p>Example:</p> <pre><element> <name>dds.transport.zrtps.library</name> <value>rtizrtps</value> </element></pre>
dds.transport.zrtps.create_function	<p>Required.</p> <p>The name of the function that will be called by <i>Connex DDS</i> to create an instance of the ZRTPS transport. The function must have the prototype NDDS_Transport_create_plugin.</p> <p>Must be set to ZRTPS_Transport_create_plugin.</p> <p>Example:</p> <pre><element> <name>dds.transport.zrtps.create_plugin</name> <value>ZRTPS_Transport_create_plugin</value> </element></pre>
dds.transport.zrtps.subtransport	<p>Required.</p> <p>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 48.</p> <p>Example:</p> <pre><element> <name>dds.transport.zrtps.subtransport</name> <value>UDPv4</value> </element></pre>
dds.transport.zrtps.aliases	<p>Required.</p> <p>Aliases used to register the ZRTPS transport plugin with the <i>DomainParticipant</i>. 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.</p> <p>Example:</p> <pre><element> <name>dds.transport.zrtps.aliases</name> <value>zrtps.udpv4</value> </element></pre>
<p><i>The following properties are optional.</i></p>	

Table 6.1 Configuration Properties for ZRTPS Plugin

Property Name	Property Value and Description
dds.transport.zrtps.compression_level	<p>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.</p> <p>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).</p> <p>Example:</p> <pre><element> <name>dds.transport.zrtps.compression_level</name> <value>9</value> </element></pre>
dds.transport.zrtps.compression_level.small_packets dds.transport.zrtps.compression_level.medium_packets dds.transport.zrtps.compression_level.large_packets	<p>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.</p> <p>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.</p> <p>Example:</p> <pre><element> <name> dds.transport.zrtps.compression_level.small_packets </name> <value>9</value> </element> <element> <name> dds.transport.zrtps.compression_level.medium_packets </name> <value>9</value> </element> <element> <name> dds.transport.zrtps.compression_level.large_packets </name> <value>9</value> </element></pre>
dds.transport.zrtps.compression_library	<p>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.</p> <p>There are several compression algorithms that can be used to compress RTPS packages:</p> <ul style="list-style-type: none"> • ZLIB_COMPRESSION: Use the Zlib algorithm from the library zlib1.dll. • BZIP2_COMPRESSION: Use the Bzip2 algorithm from the library bzip2.dll. • 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 49. • AUTOMATIC_COMPRESSION: Compress RTPS packages with all previous compression algorithms and send the smallest package. <p>Example:</p> <pre><element> <name> dds.transport.zrtps.compression_library </name> <value>AUTOMATIC_COMPRESSION</value> </element></pre>

Table 6.1 Configuration Properties for ZRTPS Plugin

Property Name	Property Value and Description
dds.transport.zrtps.compression_library.small_packets dds.transport.zrtps.compression_library.medium_packets dds.transport.zrtps.compression_library.large_packets	<p>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.</p> <p>See dds.transport.zrtps.compression_library.</p> <p>Example:</p> <pre> <element> <name> dds.transport.zrtps.compression_library.small_packets </name> <value>ZLIB_COMPRESSION</value> </element> <element> <name> dds.transport.zrtps.compression_library.medium_packets </name> <value>ZLIB_COMPRESSION</value> </element> <element> <name> dds.transport.zrtps.compression_library.large_packets </name> <value>ZLIB_COMPRESSION</value> </element> </pre>
dds.transport.zrtps.external_library	<p>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 49.</p> <p>Example:</p> <pre> <element> <name>dds.transport.zrtps.external_library</name> <value>external_library.dll</value> </element> </pre>
dds.transport.zrtps.low_mark, dds.transport.zrtps.high_mark	<p>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.</p> <p>The value for each property is an integer representing a number of bytes.</p> <p>Example:</p> <pre> <element> <name>dds.transport.zrtps.low_mark</name> <value>128</value> </element> <element> <name>dds.transport.zrtps.high_mark</name> <value>512</value> </element> </pre>
dds.transport.zrtps.verbosity	<p>The verbosity for the plugin, for debugging purposes.</p> <ul style="list-style-type: none"> • -1: Silent • 0: Exceptions only (default) • 1: Warnings • 2 and up: Debug <p>Example:</p> <pre> <element> <name>dds.transport.zrtps.verbosity</name> <value>0</value> </element> </pre>

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. *Connex DDS* 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 37](#).

6.2.1.1 Using UDPv4 as a Subtransport

To load the *Connex DDS* 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](#) below
- [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** must be the size of the entire destination buffer.
- **src_length** is the size of the compressed data in the source buffer.

Returns:

- Actual size of the uncompressed data.