Trademarks

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

Copy and Use Restrictions

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

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/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

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 sub-licenses), on a non-exclusive, perpetual, irrevocable, transferable, and worldwide basis, without any compensation, without any obligation to report on such use, and without any other restriction or obligation to you; and

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

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
Chapter 2 Paths Mentioned in Documentation ................................................................ 3

Chapter 3 Getting Started

3.1 Generate Connext Project ......................................................................................... 6
3.2 Add Name to Each DomainParticipant ...................................................................... 8
3.3 Add an RTPS Object ID to Each Endpoint ................................................................. 10
3.4 Enable LBED Plugin on Each DomainParticipant .................................................... 11
    3.4.1 Using the Builtin Discovery Plugins Mask (Recommended) ............................... 11
    3.4.2 Using the LBED Plugin Properties .................................................................... 12
3.5 Add the DDS-XML Definition of the Entities ........................................................... 13
    3.5.1 Using the USER_QOS_PROFILES.xml ............................................................ 14
    3.5.2 Using a Separate XML File .............................................................................. 21
3.6 Run the Applications ................................................................................................. 25
    3.6.1 Configure the Environment in Both Command Prompts ................................. 25
    3.6.2 Check Communication ...................................................................................... 25
    3.6.3 Check that Your Endpoints Discover Each Other Statically ............................ 27
3.7 Troubleshooting ......................................................................................................... 28
    3.7.1 System does not Recognize or Find rtiddsgen Command ............................... 28
    3.7.2 rtiddsgen Displays Error (Usually in Windows systems) ................................. 29
    3.7.3 rtiddsgen Gives Warnings ................................................................................ 29
    3.7.4 Running Publisher/Subscriber Produces Errors .............................................. 29
    3.7.5 Communication does not Occur between Publisher/Subscriber ([data: …] Messages not Displayed in Subscriber Prompt) ......................................................... 30

Chapter 4 Using LBED

4.1 Limitations ................................................................................................................ 31
4.2 LBED Properties ........................................................................................................ 32
4.3 Supported QoS

4.4 Known Issues

4.4.1 LBED not able to automatically load USER_QOS_PROFILES.xml if utilized API uses QosProvider and the file is not placed in default locations
Chapter 1 Introduction

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.

Discovery is the behind-the-scenes way in which RTI 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.

With Limited Bandwidth Endpoint Discovery (LBED), the Simple Participant Discovery phase still occurs, but the information normally gathered dynamically by Simple Endpoint Discovery
is instead configured statically in an XML file. The static definition of endpoint discovery information is why LBED can also be referred to as Static Endpoint Discovery.
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 SNDDSHOME 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 Getting Started

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 3.1: 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.
3.1 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;
   }
   ```

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.
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 3.1 Modern C++ Files Created for “ExampleLBED.idl”**

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

In the following sections, some of the generated files mentioned above will be modified for using the **Limited Bandwidth Endpoint Discovery Plugin**.
3.2 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 3.1 Generate Connext Project on page 6 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:

```xml
<!-- 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”:

```cpp
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”:

```cpp
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")
);
```
3.3 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 rticonnextdss-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.

```xml
<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 <domainParticipant_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.

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

### 3.4 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 (3.4.1 Using the Builtin Discovery Plugins Mask (Recommended) below or 3.4.2 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.

#### 3.4.1 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 (Dynamic Participant discovery, Static Endpoint discovery).

Copy and paste the following XML snippet into the <domainParticipant_qos> tag of both QoS profiles: “ExampleLBED_Publisher_Profile” and “ExampleLBED_Subscriber_Profile”: 
3.4.2 Using the LBED Plugin Properties

Using DPSE automatically configures the DomainParticipant properties and QoS in the same way as described in 3.4.2 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 4.1 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 3.4.2 Using the LBED Plugin Properties below).

3.4.2 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:

```xml
<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>
    </value>
</property>
```
3.5 Add the DDS-XML Definition of the Entities

For more information about the meaning of the properties above, their values, and other LBED properties that can be specified, see Table 4.1 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:

```xml
<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`.

### 3.5 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|LC:DISC]DDS_LBEDiscoveryPlugin_registerParticipant:The discovered DomainParticipant [guid=0x01013049,0x8484F58C,0x5EBD43BE:0x000100C7] is not in the file.
ERROR [0x01013049,0x8484F58C,0x5EBD43BE:0x000100C7{Entity=DR,MessageKind=DATA}|RECEIVE FROM 0x00000000,0x00000000,0x00000000:0x000100C2|LC:DISC]DDS_LBEDiscoveryPlugin_afterRemoteParticipantEnabled:The DomainParticipant "ExampleLBEDParticipantPublisher" 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 4.1 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 3.5.1 Using the USER_QOS_PROFILES.xml below) or by creating a separate XML file for each DomainParticipant (in 3.5.2 Using a Separate XML File on page 21). Choose one way for your application.

3.5.1 Using the USER_QOS_PROFILES.xml

If the config_file property was not specified when enabling LBED in a DomainParticipant (see 3.4.2 Using the LBED Plugin Properties on page 12), 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:
3.5.1 Using the USER_QOS_PROFILES.xml

- 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:

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

3.5.1.1 Add the types used by the endpoints

Your application Topics use the ExampleLBED type that was generated from the IDL in 3.1 Generate Connext Project on page 6. 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:

```xml
<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.

### 3.5.1.2 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:

```xml
<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 3.5.1.1 Add the types used by the endpoints on page 15, 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:

```xml
<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 3.5.1.1 Add the types used by the endpoints on page 15 (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.

### 3.5.1.3 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 endpoints 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:

```xml
<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 3.2 Add Name to Each DomainParticipant on page 8. 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 3.5.1.2 Add the Topics on page 16 (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 3.5.1.2 Add the Topics on page 16 (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.

3.5.1.4 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 4.3 Supported QoS on page 33 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:

```xml
<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_library>
</domain_participant>

<domain_participant name="ExampleLBEDParticipantSubscriber">
  <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`.

```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>
    </qos_profile>
    <qos_profile name="ExampleLBED_Subscriber_Profile">
      <domain_participant_qos>
        <protocol>
          <rtps_object_id>100</rtps_object_id>
        </protocol>
      </domain_participant_qos>
    </qos_profile>
  </qos_library>
</dds>
```
3.5.1 Using the USER_QOS_PROFILES.xml

```xml
<!-- The QoS profile used by the subscribing entities -->
<qos_profile name="ExampleLBED_Subscriber_Profile">
  <domainParticipant_qos>
    <discovery_config>
      <builtin_discovery_plugins>DPSE</builtin_discovery_plugins>
    </discovery_config>
    <participant_name>
      <name>ExampleLBEDParticipantSubscriber</name>
    </participant_name>
  </domainParticipant_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="ExampleExampleLBED" 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="ExampleExampleLBED">
        <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="ExampleExampleLBED">
        <datareader_qos
          base_name="ExampleLBED_Library::ExampleLBED_Subscriber_Profile"/>
      </data_reader>
    </subscriber>
  </domain_participant>
</domain_participant_library>
```
3.5.2 Using a Separate XML File

The `config_file` property (see Table 4.1 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 3.5.1 Using the USER_QOS_PROFILES.xml on page 14. 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.

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.
3.5.2 Using a Separate XML File

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

   
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 4.3 Supported QoS on page 33 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>
3.5.2 Using a Separate XML File

7. Since both *Topics* use the same type (ExampleLBED), copy and paste the ExampleLBED XML type definition (see 3.5.1.1 Add the types used by the endpoints on page 15) to both XML files.

8. Since both endpoints use the same *Topic* (Example ExampleLBED), copy and paste the DDS-XML snippet from 3.5.1.2 Add the Topics on page 16 (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 3.5.1.3 Add the DomainParticipants’ topology on page 17):

```
<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 3.5.1.3 Add the DomainParticipants’ topology on page 17):

```
<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 3.5.1.4 Add the required QoS profiles to the entities on page 18, 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>
</dds>
```
3.5.2 Using a Separate XML File

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

```xml
  <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="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>
        <data_writer name="DW" topic_ref="ExampleLBED">
          <datawriter_qos
            base_name="ExampleLBED_Library::ExampleLBED_Publisher_Profile_Static"/>
        </data_writer>
      </publisher>
    </domain_participant>
  </domain_participant_library>
</dds>
```
3.6 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.

3.6.1 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.

3.6.2 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:
3.6.2 Check Communication

$ ./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...
3.6.3 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":

   ```xml
   <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:
3.7 Troubleshooting

3.7.1 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
3.7.2 rtiddgen 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 rtiddgen with the -ppDisable option.

3.7.3 rtiddgen 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.

3.7.4 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 3.6.1 Configure the Environment in Both Command Prompts on page 25.

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 3.6.1 Configure the Environment in Both Command Prompts on page 25.

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: 4.4.1 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 36.
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.

### 3.7.5 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 3.5 Add the DDS-XML Definition of the Entities on page 13.
- 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.
Chapter 4 Using LBED

4.1 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:

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

- Only the Connex 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 4.3 Supported QoS on page 33), 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 4.3 Supported QoS on the next page and 4.4 Known Issues on page 36 for further information about what is and is not supported in LBED.

4.2 LBED Properties

Table 4.1 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 4.1 LBED Configuration Properties for Connext

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value and Description</th>
</tr>
</thead>
</table>
| dds.discovery.endpoint.load_plugins | Required only when the plugin is enabled using the LBED Properties (see 3.4.2 Using the LBED Plugin Properties on page 12). String indicating the prefix name of the plugin that will be loaded by Connext. 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:  
<element>  
  <name>dds.discovery.endpoint.load_plugins</name>  
  <value>dds.discovery.endpoint.lbediscovery</value>  
</element> |
| dds.discovery.endpoint.<string>.library | Required only when the plugin is enabled using the LBED Properties (see 3.4.2 Using the LBED Plugin Properties on page 12). 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. Set the value to `rtlibdis`. Example:  
<element>  
  <name>dds.discovery.endpoint.lbediscovery.library</name>  
  <value>rtlibdis</value>  
</element> |
| dds.discovery.endpoint.<string>.create_function | Required only when the plugin is enabled using the LBED Properties (see 3.4.2 Using the LBED Plugin Properties on page 12). The name of the function that will be called by Connext to create an instance of the LBED plugin. Set the value to `DDS_LBEDiscoveryPlugin_create`. Example:  
<element>  
  <name>dds.discovery.endpoint.lbediscovery.create_function</name>  
  <value>DDS_LBEDiscoveryPlugin_create</value>  
</element> |
### 4.3 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.

#### Table 4.1 LBED Configuration Properties for Connext

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value and Description</th>
</tr>
</thead>
</table>
| dds.discovery.endpoint.<string>.config_file | Optional.  
The absolute or relative path to the file that contains the DDS-XML static definition of the endpoints that must be discovered.  
Example:  
```xml  
<element>  
  <name>dds.discovery.endpoint.lbediscovery.config_file</name>  
  <value>LBEDSubscriber.xml</value>  
</element>  
```  
Use this property only if:  
- 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 3.5.2 Using a Separate XML File on page 38.  
- 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 4.4 Known Issues on page 30 for further information. |
| dds.discovery.endpoint.<string>.verbosity | Optional.  
The verbosity for the plugin, for debugging purposes.  
- -1: Silent  
- 0: Exceptions only (default)  
- 1: Warnings  
- 2 and up: Debug  
Example:  
```xml  
<element>  
  <name>dds.discovery.endpoint.lbediscovery.verbosity</name>  
  <value>1</value>  
</element>  
```  
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.  
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:  
- VALIDATION_ACTION_EXCEPTION: validate the properties. Upon failure, log errors and fail.  
- VALIDATION_ACTION_SKIP: skip validation.  
- VALIDATION_ACTION_WARNING: validate the properties. Upon failure, log warnings and do not fail.  
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. |
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
4.3 Supported QoS

- 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.
4.4 Known Issues

4.4.1 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:

```xml
<!-- USER_QOS_PROFILES.xml -->
<qos_profile name="...">
  <domain_participant_qos>
    <discovery_config>
```

4.4.1 LBED not able to automatically load USER_QOS_PROFILES.xml if utilized API uses

```xml
<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>
```