Contents

1 Installation
  1.1 Introduction ................................................................................................................ 1-1
  1.2 Installing .................................................................................................................. 1-1
    1.2.1 Installing RTI DDS Toolkit Support Files on a Target........................................... 1-4
  1.3 Verifying Installation ............................................................................................... 1-6
    1.3.1 LabVIEW Functions Palette .................................................................................... 1-7
    1.3.2 LabVIEW Controls Palette.................................................................................... 1-8
  1.4 Upgrading ............................................................................................................... 1-9
    1.4.1 Additional Steps when Upgrading from a Release Older than 2.0.0.104 ................... 1-9
  1.5 Uninstalling ........................................................................................................... 1-10
    1.5.1 Uninstalling RTI DDS Toolkit Support Files from LabVIEW RT Targets ............... 1-11
  1.6 LabVIEW Examples ................................................................................................ 1-12
  1.7 Product Support ..................................................................................................... 1-13

2 Communication Models
  2.1 Publish/Subscribe – A Simple Analogy ........................................................................ 2-2
  2.2 The DDS Paradigm .................................................................................................... 2-3
  2.3 Quality of Service (QoS) ............................................................................................. 2-4
  2.4 DDS—Example Application ......................................................................................... 2-5

3 A Simple Read/Write Example
  3.1 Publishing a String in DDS ........................................................................................ 3-2
  3.2 Subscribing to a String in DDS .................................................................................. 3-2
  3.3 What is Happening? ..................................................................................................... 3-3
  3.4 Usage Notes ................................................................................................................ 3-5
    3.4.1 Preventing ‘Application Failed to Start’ Error when Opening Example VIs ............ 3-5
    3.4.2 Communicating Unbounded Entities....................................................................... 3-5
    3.4.3 Preventing ‘Type Code Incorrect’ Error when Working with Arrays.................... 3-5
    3.4.4 Troubleshooting with Ping and Spy........................................................................... 3-6

4 Tutorial
  4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) .................. 4-2
    4.1.1 Developing a VI to Publish Simple Data (Numeric) .................................................. 4-2
      4.1.1.1 Create a Writer Object to Publish a Numeric (DBL) ........................................... 4-2
      4.1.1.2 Publish a Numeric (DBL).................................................................................. 4-3
      4.1.1.3 Release the Writer Object.................................................................................. 4-4
    4.1.2 Creating a VI to Subscribe to Simple Data (Numeric) ........................................... 4-4
      4.1.2.1 Create a Reader Object to Subscribe to a Numeric (DBL)................................. 4-4
5 Loading Quality of Service Profiles

6 Advanced Concepts and Settings

6.1 Default Configuration: DDS Entities Created by ‘Simple Create’ SubVIs

6.2 Types with a Specific String Size

6.3 RTI DDS ComplexType Generator

6.4 Configuring Advanced Writer Settings

6.5 Configuring Advanced Reader Settings
6.6 Working with Instance State Kind ............................................................................................................. 6-8
6.6.1 Write, Dispose or Unregister .................................................................................................................. 6-9
6.6.2 Reading Instance State Kind .................................................................................................................. 6-9
6.7 Debugging an RTI Connext DDS LabVIEW Application ........................................................................ 6-11
6.7.1 Using Administration Panel (for Windows Systems only) ..................................................................... 6-12
   6.7.1.1 Configuration Section ...................................................................................................................... 6-14
   6.7.1.2 DDS State Info ................................................................................................................................. 6-16
   6.7.1.3 Debugging Table ............................................................................................................................. 6-16
6.7.2 Debugging SubVIs on Real-Time Targets and Windows Systems ......................................................... 6-16
   6.7.2.1 Get Configuration Parameters .................................................................................................... 6-17
   6.7.2.2 Set Configuration Parameters ....................................................................................................... 6-17
   6.7.2.3 Get DL Configuration Parameters ................................................................................................. 6-17
   6.7.2.4 Configure Distributed Logger ........................................................................................................ 6-18
6.7.2.5 DDS State Info ................................................................................................................................. 6-18
6.7.2.6 Reading Logged Messages ................................................................................................................ 6-18
6.7.3 Logging Messages from LabVIEW ........................................................................................................ 6-19
6.8 Enabling Security ...................................................................................................................................... 6-20
   6.8.1 Managing Custom Security Profiles with the Security Panel (Windows Systems) ......................... 6-21
      6.8.1.1 Creating Custom Security Profiles .............................................................................................. 6-24
      6.8.1.2 Deleting Custom Security Profiles .............................................................................................. 6-24
      6.8.1.3 Load Custom Security Profile Values ........................................................................................... 6-25
   6.8.2 Managing Custom Security Profiles with SubVIs ........................................................................ 6-25
      6.8.2.1 Creating Custom Security Profiles .............................................................................................. 6-25
      6.8.2.2 Deleting Custom Security Profiles .............................................................................................. 6-25
      6.8.2.3 Getting Custom Security Profiles List .......................................................................................... 6-26
      6.8.2.4 Get Security Profiles Values ........................................................................................................ 6-26
   6.8.3 Creating DomainParticipants using a Custom Security Profile ......................................................... 6-26
6.9 Advanced Filtering of Data—ContentFilteredTopics .............................................................................. 6-27
   6.9.1 Configuring ContentFilteredTopics .................................................................................................. 6-28

A VI Descriptions
A.1 Controls Palette Types ................................................................................................................................. A-1
A.2 Functions Palette ......................................................................................................................................... A-3
   A.2.1 Writer .................................................................................................................................................. A-3
   A.2.2 Reader ................................................................................................................................................ A-4
   A.2.3 Tools .................................................................................................................................................. A-6
      A.2.3.1 DDS Debugging SubPalette ........................................................................................................ A-7
   A.2.4 DDS Security Subpalette ................................................................................................................... A-9

B Creation and Release of DDS Entities

C Supported Data Types and Corresponding IDL
   C.1 Corresponding IDL for Complex Data Types ......................................................................................... C-4
      C.1.1 Clusters .......................................................................................................................................... C-4
      C.1.2 Enums ........................................................................................................................................... C-5

D File Folders Installed within LabVIEW
   D.1 File Folders on Windows Systems ......................................................................................................... D-1
   D.2 File Folders on NI Linux Targets ......................................................................................................... D-2
E Troubleshooting
E.1 Enabling Debugging Mode ................................................................. E-1
E.2 Error Codes and Possible Solutions .................................................. E-1
E.3 Running without an Active Network Interface .................................... E-9
E.4 Error Installing RTI DDS Toolkit RT Support ..................................... E-10
E.5 Error Using Custom Security Profiles ................................................ E-10
E.6 Error Generating ComplexType VIs ................................................... E-11
E.7 RT Device Hangs when Modifying QoS Profiles ................................. E-11
Chapter 1  Installation

1.1  Introduction

Developing heterogeneous distributed systems is a complex challenge. Individual subsystems are often developed by independent teams, third parties, and legacy systems. These complexities can be substantially reduced by leveraging the combined power of RTI® Connext® DDS and National Instruments® LabVIEW™.

By using LabVIEW and Connext DDS together, you can develop advanced and unique system architectures to simplify system integration, data communication, network bandwidth management, and redundancy.

This document will help you install and get started with RTI DDS Toolkit. The instructions assume you are already familiar with the basics of using LabVIEW.

1.2  Installing

Notes:

- If you are upgrading RTI DDS Toolkit, skip to Upgrading (Section 1.4).
- You need administrator privileges to install the toolkit and to make it available to install in RT targets.
Installing

To Install RTI DDS Toolkit:

1. Verify you have a supported version of LabVIEW already installed (see the Release Notes for supported versions).
2. Login with administrator privileges.
3. Install the JKI LabVIEW VI Package Manager (VIPM) if you have not done so already (available here: http://jki.net/vipm/download). It is typically installed in C:\Program Files (x86)\JKI\VI Package Manager.
4. Make sure LabVIEW is not running.
5. Launch the VIPM in elevated mode.
6. Look for ‘RTI DDS’ in the search menu and double-click on RTI DDS Toolkit.

Note: In LabVIEW 2017, you will find a shortcut from the block diagram: select Data Communication, RTI DDS Toolkit, Install, as seen below:
7. Install RTI DDS Toolkit:
   a. Select the LabVIEW version for which you want to install RTI DDS Toolkit.
   
   ![Image of Package Information]

   If you have more than one version of LabVIEW installed, you will be able to select a version from a drop-down list.

   b. Select Install.

8. The VIPM will start the installation process and display a window similar to the one below. You need to accept the license to proceed.

   ![Image of License Agreements]

   **Note:** When running the VIPM for the first time, the VIPM will test the connection to LabVIEW and display the default port for LabVIEW. Select Test and allow the test to complete.

   During this step, the VIPM launches the LabVIEW version selected for the RTI DDS Toolkit installation. The LabVIEW application will appear in the Windows Task Bar at the bottom of your screen. You may need to open the LabVIEW application from the Task Bar and select Launch LabVIEW before the VIPM test times out.

9. If offered, select Finish when the installation is complete.
1.2.1 Installing RTI DDS Toolkit Support Files on a Target

Note: Your target will be rebooted as part of the installation process.

To install Real-Time target support for RTI DDS Toolkit:

RTI DDS Toolkit support files allow you to deploy VIs using RTI DDS Toolkit into your target. The following instructions assume you have JKI VIPM and LabVIEW installed. The following instructions assume you have installed successfully the RTI DDS Toolkit which is explained in Installing (Section 1.2).

1. Launch NI MAX (Measurement & Automation Explorer).
4. Login with administration privileges in your target.
5. In the LabVIEW Real-Time Software Wizard, select **Custom software installation**. A dialog will ask if you are sure you want to install customized software. Click **yes**.

6. Navigate to the **RTI DDS Toolkit** feature, click on the icon to the left of the name and select **Install the feature**.

   ![Image of LabVIEW Real-Time Software Wizard]

   **Note:** If you are upgrading to a newer release, make sure you select the newest version from the drop-down list on the right side (**Available version(s) on the host**).

7. Click **Next** and verify **RTI DDS Toolkit** is selected to be installed.

8. Click **Next**. The installation will start, then the target will automatically reboot.
After installation, **RTI DDS Toolkit** will appear in the installed Software list of your target.

### 1.3 Verifying Installation

1. Launch LabVIEW.
2. Select **File, New VI**.
3. From the Block Diagram’s **View** menu, open the **Functions Palette**. From this palette, select the down arrows at the bottom. Select **Data Communication** and verify that you see **RTI DDS Toolkit**.

For details, see **LabVIEW Functions Palette** (Section 1.3.1).
4. From the Front Panel’s View menu, open the Controls Palette. From this palette, select the down arrows at the bottom. Select RTI DDS Toolkit and verify that you see RTI DDS Toolkit’s controls.

For details, see LabVIEW Controls Palette (Section 1.3.2).

See also: Appendix D: File Folders Installed within LabVIEW.

1.3.1 LabVIEW Functions Palette

RTI DDS Toolkit adds the following to the Data Communication section of the Block Diagram’s Functions Palette:

- **RTI DDS Toolkit**
  - Writer
    - Simple Create Writer
    - Advanced Create Writer
    - Write
    - Release Writer
    - Set Writer QoS
  - Reader
    - Simple Create Reader
    - Advanced Create Reader
    - Read
    - Release Reader
    - Set Reader QoS
• **Tools**
  - DDS Release Unused Entities
  - DDS Time to LV Time
  - DDS Debugging
    - Get Configuration Parameters
    - Set Configuration Parameters
    - Get DL Configuration Parameters
    - Configure Distributed Logger
    - Get DDS State
    - Read One Logged Message
    - Log New Message

• **DDS Security**
  - Create Custom Security Profile
  - Delete Custom Security Profile
  - Get Custom Security Profiles List
  - Get Security Profile Values

1.3.2 **LabVIEW Controls Palette**

*RTI DDS Toolkit* adds the following to the Addons section of the Front Panel’s Controls Palette:

- **RTI DDS Toolkit**
  - RTI DDS Advanced Reader Configuration
  - RTI DDS Advanced Writer Configuration
  - DDS Sample Info
  - DDS State Info
  - RTI DDS Security Settings
  - RTI DDS ContentFilteredTopic Info
  - RTI DDS Filter Level
  - RTI DDS Write Sample Kind
1.4 Upgrading

If you have already installed RTI DDS Toolkit and are upgrading to a newer release:

1. Login with administrator privileges.
2. Ensure that LabVIEW is not running.
3. Launch the VIPM in elevated mode, then:
   a. Look for the 'RTI DDS Toolkit' latest version in the search bar.
   b. (or) Select File, Open Package File(s) and open the latest RTI DDS Toolkit .vip file.
4. Upgrade RTI DDS Toolkit:
   a. Select the LabVIEW version for which you want to upgrade RTI DDS Toolkit.
      • If you have more than one version of LabVIEW installed, you will be able to select the LabVIEW version from the LabVIEW version drop down list.
      • The VIPM allows you to view all versions of RTI DDS Toolkit available to your system by selecting *Browse All Versions in the lower-left corner.
   b. Select Upgrade.
5. The VIPM will start the installation process. Select Continue to proceed.
6. If prompted, select Finish when the installation is complete.

1.4.1 Additional Steps when Upgrading from a Release Older than 2.0.0.104

If you are upgrading from a version older than 2.0.0.104, you must follow these steps to upgrade your VIs to the newer version. Follow these instructions after upgrading the toolkit.

☐ The create Reader/Writer subVIs have been removed. Use the create simple/advanced Reader/Writer subVIs instead. These VIs are included in the RTI DDS Toolkit/Reader and RTI DDS Toolkit/Writer subpalettes.
The Complex-Type Templates are no longer supported. Therefore, that subpalette has been removed. Instead, you can use RTI DDS ComplexType Generator (Section 6.3).

The ForceNewDomainParticipant? flag from the Advanced Reader/Writer Configuration has been deleted. If these clusters are not updated automatically, they need to be updated manually. To do so, go to the Front Panel, right-click on the old cluster, select Replace, RTI DDS Toolkit and choose the new cluster.

1.5 Uninstalling

To uninstall RTI DDS Toolkit:

1. Login with administrator privileges.
2. Ensure that LabVIEW is not running.
3. Launch the VIPM in elevated mode, then:
   a. Scroll down to locate RTI DDS Toolkit.
   b. Double-click on RTI DDS Toolkit to open the Package Information screen.
4. Select the LabVIEW version you want to work with from the LabVIEW version drop-down list.

   **Note:** The VIPM allows you to view all versions of *RTI DDS Toolkit* available to your system by selecting *Browse All Versions* in the bottom left-hand corner.

5. Select **Uninstall**.

6. Select **Continue**.

7. If offered, select **Finish** when the VIPM finishes uninstalling *RTI DDS Toolkit*.

### 1.5.1 Uninstalling RTI DDS Toolkit Support Files from LabVIEW RT Targets

To uninstall LabVIEW RT support files for *RTI DDS Toolkit*:

1. Make sure no LabVIEW application is using the libraries on the target.

2. Launch NI MAX.


4. Go to **Software** and click on **Add/Remove Software** to launch the LabVIEW Real-Time Software Wizard.

5. Login with administrator privileges to your target.

6. In the LabVIEW Real-Time Software Wizard, select **Custom software installation**. A dialog will ask if you are sure you want to install customized software. Click **yes**.

7. Navigate to the **RTI DDS Toolkit** feature, click on the icon to the left of the name and select **Uninstall the feature**.

8. Click **Next** and verify **RTI DDS Toolkit** is selected to be uninstalled.

9. Click **Next**. The uninstallation will start, then the target will automatically reboot.
1.6 LabVIEW Examples

RTI DDS Toolkit includes several examples which are used in later chapters. To access these examples:

1. Select the LabVIEW Help menu.
2. Select Find Examples…
3. In the Browse tab, select the radio button to browse according to Directory Structure:
4. Scroll down and open the RTI DDS Toolkit folder.

You will find the following examples:

- **ClusterDemo**: Shows how to handle complex types (such as clusters). It was created by following the lessons in Chapter 4: Tutorial:
  - Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2)
  - Lesson 3—Filtering Data (Section 4.3)
  - Lesson 4—Reading Only New Samples (Section 4.4)
- **ContentFilteredTopicDemo**: shows how to filter data using a ContentFilteredTopic. It has been created by following Filtering Data Using ContentFilteredTopics (Section 4.3.2).
- **cRIOProject**: Shows how to use RTI DDS Toolkit on a cRIO 9068. It is explained in Lesson 9—Using RTI DDS Toolkit on NI Targets (cRIO-9068 Example) (Section 4.9).
- **LogMessagesDemo**: Shows how to log debugging messages into the internal queue. It was created by following Logging Messages Manually (Section 4.8.1.1).
- **MonitoringDemo**: Uses a QoS profile that enables RTI Monitoring Library. It was created by following Adapting a VI to Use RTI Monitoring Library (Section 4.8.2).
- **NumberDemo**: Shows how to read and write a simple type (such as a numeric one). It was created by following Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) (Section 4.1).
- **ReadAllDemo**: Shows how to read all available data by calling the Read function several times and storing the data in an array without adding already existing samples. It is explained in Lesson 7—Reading All Samples (Reliable Communication) (Section 4.7).
- **SecurityShapesDemo**: This example will show how to use different security profiles and how they behave depending on the permissions they have. It is explained in Lesson 10—Using Security with RTI DDS Toolkit (Section 4.10).
- **ShapesDemo**: Shows how to publish and subscribe to an already existing DDS application: RTI Shapes Demo. It is explained in Lesson 5—Using Keyed Types (RTI Shapes Demo) (Section 4.5).
- **StringsDemo**: Shows how to write a string. It is explained in Chapter 3: A Simple Read/Write Example.

**Note:** If you see an error after opening one of the examples (such as “This application has failed to start because its side by side configuration is incorrect”), see Section 3.4.3.
1.7 Product Support

For technical support or questions about RTI DDS Toolkit, please visit the National Instrument User Group “RTI DDS Toolkit for LabVIEW Support” (https://forums.ni.com/t5/RTI-DDS-Toolkit-for-LabVIEW/gp-p/5344) or the RTI Community portal (http://community.rti.com).

If you have an RTI support subscription, please contact support@rti.com. If you do not have an RTI support subscription, you can acquire one by contacting labview@rti.com.
Chapter 2  Communication Models

This section provides an overview of middleware communication paradigms, including publish-subscribe, along with details of the OMG Data Distribution Service (DDS) standard.

Software applications are becoming increasingly distributed. A node in a distributed system must access the right data, know where to send it, and deliver it to the right place at the right time. Simplifying the access to this data would enable a whole new class of distributed applications. The challenge, especially in mission-critical and time-critical networks, is to quickly access and disseminate information to many nodes.

Three major middleware communication paradigms have emerged to meet this need:

- Client/Server
- Message passing
- Publish/Subscribe

**Client/Server** is fundamentally a many-to-one design that works well for systems with centralized information, such as databases, transaction processing systems, and central file servers. However, if multiple nodes generate information, client/server architectures require all the information be sent to the server for later redistribution to the clients, resulting in inefficient client-to-client communication.

The central server is a potential bottleneck and single-point of failure. It also adds inefficiencies and unknown delay (and therefore indeterminism) to the system, because the receiving client does not know when it has a message waiting, so it has to keep polling periodically.

**Message Passing** architectures work by implementing queues of messages. Processes can create queues, send messages, and service messages that arrive. Message passing makes it easier to exchange information between many nodes in the system. However, applications remain coupled. Each message placed in a queue goes to a single consumer and the addition of new consumers impacts the network.

In practice, applications find data indirectly by targeting specific sources (e.g., by process ID, "channel", or queue name) on specific nodes. So this architecture does not address how applications know the location of a process/channel, what happens if that process/channel does not exist, etc. The application must determine where to get data, where to send it, and when to perform the transaction. A message-passing architecture provides a model for the transfer of data, but no model for the data itself.

**Publish/Subscribe** decouples the producers and consumers of the information. Producer publishes data they have and consumers subscribe to data based on their interests. The publish/subscribe middleware infrastructure is responsible for delivering each message published to all interested consumers. Applications remain decoupled because the presence of new consumers...
does not perturb existing consumers. Existing consumer’s requirements are met, regardless of how many other consumers subscribe to the same data.

The fundamental communications model implies both discovery (i.e., what data should be sent) and delivery (i.e., when and where to send the data). This design mirrors time-critical and mission-critical information delivery systems in everyday life (e.g., television, radio, magazines and newspapers). The publish/subscribe network architecture is excellent at distributing large quantities of time-critical information quickly, even in the presence of unreliable delivery mechanisms.

The publish/subscribe architecture maps well to high-performance and real-time communication challenges. Finding the right data becomes straightforward; nodes just declare their interest once and the middleware handles all the details of the network and delivery. Sending the data quickly is also inherent; publishers send data when the data is available. Publish/subscribe is highly efficient because the data flows directly from source (publisher) to destination (subscriber) without requiring intermediate servers, brokers, or daemons. Multiple sources and destinations are easily defined within the model, providing inherent redundancy and fault tolerance.

Data-Centric Publish/Subscribe (DCPS) middleware, such as the OMG Data Distribution Service (DDS), defines a data model on top of the publish/subscribe infrastructure, allowing the data to be structured. The schema of the data being published is declared by the application and known to the middleware. Similar to the relational model in databases, each data type (a DDS Topic) has an associated schema and a set of attributes that identify the ‘key’ for that Topic. Data published on that Topic is understood by the middleware, allowing advanced capabilities such as content-based filtering, last value (or history) caching, and applying fine-grained Quality of Service (QoS) separately for each data-object written to the Topic.

In summary,

- Client/server middleware is best for centralized data designs and for systems where the dominant communication pattern is request-reply, such as file servers and transaction systems.
- Message passing, with its "send that there" semantics, maps well to systems with clear and simple data-flow requirements, and requires the application to discover where data resides.
- Publish/subscribe, by providing both discovery and messaging, decouples the producers and consumers effectively. DCPS middleware provides publish/subscribe services to an application-defined data-model, allowing fine-grained control of QoS, enabling the infrastructure to do smart-caching of the information and provide content and time filtering at the source and destination. The data-centric architecture provides the best decoupling between application components and is best suited for time-critical and mission critical distributed applications.

## 2.1 Publish/Subscribe – A Simple Analogy

The publish/subscribe communications model is analogous to that of a traditional magazine or newspaper business model. A Topic represents the kind of publication (data or information), for example “Newspaper” or “Magazine”. If we use the Newspaper as the model, the Key is used to identify each different news corporation (“New York Times”, “San Francisco Chronicle”, “La Strada”, “Le Monde”, etc.). The type specifies the format of the information (how it is encoded). The user data is the contents (text and graphics) of each sample (weekly or daily issues). The middleware is the distribution service (US Postal Service or a paper delivery service) that deliv-
ers the publication from where it is created (a printing house) to the individual subscribers (people’s homes). This analogy is illustrated in Figure 2.1.

Note that by subscribing to a publication, subscribers are requesting current and future samples of that publication, so that as new samples are published, they are delivered without having to submit another request for data. By specifying a content-filter on the value of the Key (the periodical name in this case) a subscriber may indicate he only wants certain periodicals (e.g., yes to the “New York Times” and “La Strada”, but no to others). Content filters could also select based on other attributes in the data (e.g., select the ones written in a specific language, or coming from a specific region). Time-based filters can be used to request only a subset of the samples (e.g., only the Sunday edition).

Figure 2.1 An Example of Publish-Subscribe

The publish/subscribe model is analogous to publishing magazines or newspapers. The Publisher sends samples of a particular Topic to all Subscribers of that Topic.

In this example, Quality of Service (QoS) parameters can be linked to delivery requirements; only deliver the Sunday edition, the paper must be delivered by 7:00am, the paper must be in the mailbox or on the porch, or delivered by certified mail with the subscriber signing receipt of delivery.

QoS parameters specify how, where, and when the data is to be delivered, controlling not only transport-level delivery properties, but also application-level concepts of fault tolerance, ordering, and reliability.

2.2 The DDS Paradigm

The Object Management Group (OMG) Data Distribution Service (DDS) standard the comprehensive specification available for publish/subscribe data-centric designs. The DDS publish/subscribe model connects anonymous information producers (publishers) with information consumers (subscribers). The overall distributed application is composed of processes called "Participants," each running in a separate address space, and often on different computer or system nodes. A Participant may simultaneously publish and subscribe to typed data-streams identified by a string name, these streams are called Topics in DDS. The model allows publishers and subscribers to present type-safe interfaces to the application.

DDS defines a communications relationship between publishers and subscribers. The communications are decoupled in space (nodes can be anywhere—same node, a local node, or a geographically remote node), time (delivery may be immediate or controlled), and flow (delivery may be reliable with a controlled bandwidth). To increase scalability, Topics may contain multiple independent data channels identified by "Keys." This allows system nodes to subscribe too many, possibly thousands, of similar data streams with a single subscription. When the data
arrives, the middleware can cache and sort data using the Key and deliver it for efficient processing.

Additionally, DDS is fundamentally designed to work over unreliable transports, such as UDP, wireless, or disadvantaged networks without the requirement for central servers or special nodes. Direct, peer-to-peer communications, and support for reliable multicasting, enable a highly efficient data distribution model.

2.3 Quality of Service (QoS)

Fine-grained control over QoS is a powerful feature of DDS. Each publisher/subscriber pair can establish independent QoS agreements. Thus, DDS designs can support extremely sophisticated and flexible data-flow requirements.

QoS parameters control most aspects of the DDS paradigm and the underlying communication mechanisms. Many QoS parameters are implemented as “contracts” between publishers and subscribers; publishers offer and subscribers request levels of service. The middleware is responsible for determining if the offerer can satisfy the subscriber’s request, thereby establishing communication, or indicating an incompatibility error. Ensuring that publish/subscribe pairs meet the level-of-service contracts guarantees predictable operation. Information about some common QoS parameters is presented below.

- **Deadline:** Periodic publishers can indicate the speed at which they can publish by offering guaranteed update deadlines. By setting a deadline, a compliant publisher promises to send a new update on each key at a minimum rate. Subscribers may then request data at that or any slower rate.

- **Reliability:** Publishers may offer levels of reliability, parameterized by the number of past issues they can store for the purpose of retrying transmissions. Subscribers may then request differing levels of reliable delivery, ranging from fast-but-unreliable “best effort” to highly reliable in-order delivery. This provides per-data stream reliability control.

- **Strength:** The middleware can automatically arbitrate between multiple publishers of the same data with a parameter called “strength.” For each keyed data-object the subscriber receives data only from the strongest active publisher of that key. This provides automatic failover; if a strong publisher fails, all subscribers immediately receive updates from the backup (weaker) publishers.

- **Durability:** Publishers can declare “durability,” a parameter that determines how long previously published data is saved. Late-joining subscribers to durable publications can then be updated with a snapshot containing the most current set of values for each Key. Other QoS parameters control when the middleware detects nodes that have failed, suggest latency budgets, set delivery order, attach user data, prioritize messages, set resource utilization limits, partition the system into namespaces, and more. The DDS QoS facilities offer extensive flexibility and communications control.

**RTI DDS Toolkit** includes a set of predefined QoS profiles. These profiles are embedded in **RTI DDS Toolkit** and cannot be modified. You can inherit from them. For your convenience, you can find an XML file that shows you these profiles in **C:/Program Files/National Instruments/LabVIEW 20xx/vi.lib/_RTI DDS Toolkit_interal_deps/RTI_LABVIEW_CONFIG/documentationONLY.xml** (where 20xx depends on your LabVIEW version). As the filename suggests,
2.4 DDS—Example Application

An air traffic control system provides sufficient details and requirements for as example application. An air traffic control system may monitor and direct all flights over an entire continent. The data distributed in such a system is in the form of aircraft tracks, which provides positional information (e.g., course, speed, etc.) about an airplane. Components of an air traffic control system would include radar systems, airplanes and air traffic control centers that provide current flight status information through real-time displays.

Managing the correct distribution of data in such a system can be complex. Each radar system can track many different airplanes, and each airplane may be tracked by more than one radar system. Real-time access to this information is needed for displays at air-traffic control centers so that air traffic controllers can make informed decisions. Air traffic controllers in the north-east may only want aircraft track information in their area, so only a subset of data needs to be provided to them. Based on current local conditions (e.g., air traffic, weather, etc.) air traffic controllers may issue flight plan updates to the pilot in order to route around inclement weather and other airplanes. Though a specific plane does not need flight plans from all other airplanes, it would be useful to have information about planes in the immediate vicinity.

Defining the air traffic control system in terms of publishers, subscribers and QoS parameters reveals that DDS is a natural fit to address this data distribution problem. Each radar system can be thought of as a publisher that publishes the "tracks" Topic which describes an airplane’s positional information. Each airplane that the radar system is tracking can be thought of as an "instance" of the track Topic identified by a unique Key attribute (e.g., the Airline name and flight number). The real-time controller displays subscribe to the tracks Topic and publish “flight plan” Topic updates back to the specific airplane. QoS parameters can be used to manage and control deterministic behaviors and fault tolerance capabilities of the system.
Chapter 3  A Simple Read/Write Example

The best way to learn about RTI DDS Toolkit is to begin building example applications. The following example VIs provide a quick introduction to the capabilities:

- RTI Connext DDS Read String.vi
- RTI Connext DDS Write String.vi

After reading this chapter, we recommend completing the lessons in Chapter 4: Tutorial for a more in-depth look at the capabilities of RTI DDS Toolkit.

Note: The instructions for this example assume you are already familiar with LabVIEW.

Before continuing, please make sure you have the following software installed:

- LabVIEW (32-bit) for Windows (see the Release Notes for supported versions)
- RTI DDS Toolkit

If you are using a computer that does not have an active network interface, see Running without an Active Network Interface (Section E.3).

We will start with the StringsDemo example VIs. To access the examples:

1. Launch LabVIEW.
2. From the LabVIEW Help menu, select Find Examples….
3. Select the Browse according to: Directory Structure radio button.
4. Scroll down and open the RTI DDS Toolkit folder.
5. Open the StringsDemo folder.

Notes:

- If you see an error after opening one of the examples (such as “This application has failed to start because its side by side configuration is incorrect”), see Section 3.4.4.
- If the example VI seems blocked (the stop button toggles, data does not transfer, etc.), you may have a linking issue in the VI. This issue is very likely for LabVIEW 2010 users. Section 3.4.1 explains how to resolve this.
3.1 Publishing a String in DDS

1. Open the RTI Connext DDS Write String.vi by double-clicking on it in the NI Example Finder (select Help, Find Examples...).
2. Click the Run button in the LabVIEW toolbar.
3. From the LabVIEW Front Panel, enter some text (such as Hello DDS) in the Text field and click the Enter Text button in the LabVIEW toolbar.

You are now writing (publishing) the string using DDS. Next we will read it from the RTI Connext DDS Read String.vi.

3.2 Subscribing to a String in DDS

1. Open the RTI Connext DDS Read String.vi by double-clicking on it in the NI Example Finder (select Help, Find Examples...).
2. Click on the Run button in the LabVIEW toolbar.
3. Verify that it is reading the same string that is being published from the RTI Connext DDS Write String.vi.

While both VIs are running, verify that if you change the text in the Text control of the RTI Connext DDS Write String.vi, you will read the new text in the RTI Connext DDS Read String.vi. Remember to use the LabVIEW Enter Text button in the toolbar (rather than pressing Enter or Return on your keyboard).

Note: Under the DDS publish/subscribe paradigm, knowing the location of the distributed applications is handled by the middleware. In this example, we are running both the RTI Connext DDS Write String.vi and the RTI Connext DDS Read String.vi on the same computer, using the Shared Memory transport for inter-application communication. However, if you were to run these examples on different computers (with a functional LAN connection), DDS would automatically handle the communication across the network.

3.3 What is Happening?

To better understand how this demonstration is implemented, let’s review the code for these two VIs:

- Publisher side

  The RTI Connext DDS Write String.vi uses three RTI DDS Toolkit subVIs:
  
  - **Simple Create Writer**: Creates a Writer object for text (strings) and initializes it according to the VI configuration parameters.
  
  - **Write**: Receives as input the reference from the Writer object (Create Writer) and the text to be published (the Text control). It will continue publishing the text within a LabVIEW loop until an error occurs or the Stop Writing control is pressed.
**Release Writer:** When the *Stop Writing* control is pressed, the *Release Writer* subVI will execute and release the *Writer* object.

For details on these subVIs, see **Writer (Section A.2.1).**

If you open the Block Diagram (in the RTI Connext DDS Write String Example window, select **Window, Show Block Diagram**), it will look like this:

![Block Diagram for RTI Connext DDS Write String](image)

- **Subscriber side**

  The **RTI Connext DDS Read String.vi** uses three **RTI DDS Toolkit subVIs:**
  - **Simple Create Reader:** Creates a *Reader* object for text (strings) and initializes it according to the VI configuration parameters.
  - **Read:** Receives as input the reference from the *Reader* object (*Create Reader*). Outputs the *Text* indicator. It continues subscribing to the text within a LabVIEW loop until an error occurs or the *Stop Reading* control is pressed.
  - **Release Reader:** When *Stop Reading* control is pressed, the *Release Reader* subVI will execute and release the *Reader* object.

  For details on these subVIs, see **Reader (Section A.2.2).**

  If you open the Block Diagram (in the RTI Connext DDS Read String Example window, select **Window, Show Block Diagram**), it will look like this:

![Block Diagram for RTI Connext DDS Read String](image)
3.4 Usage Notes

3.4.1 Preventing ‘Application Failed to Start’ Error when Opening Example VIs

If you see an error when LabVIEW tries to load the *RTI DDS Toolkit* DLL (such as “This application has failed to start because its side by side configuration is incorrect.”) after opening any of the example VIs, you need to install the *Microsoft Visual C++ 2008 Redistributable Package* (x86). This package provides the run-time components of the Visual C++ Libraries that are required to run applications developed with Visual C++ on a computer that does not have the Visual C++ 2008 development environment. You can download this package from [http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=29](http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=29).

3.4.2 Communicating Unbounded Entities

By default, strings in *RTI DDS Toolkit* are bounded so their maximum length is 1024 characters. However, if you set the Advanced Reader/Writer Configuration flag `forceUnboundedString` to `true`, they are created with a length equivalent to the maximum integer (2,147,483,647) (see Lesson 6—Used Nested and Multiple Keys (Section 4.6)). Despite that, DDS only sends the actual data the string contains, automatically reducing the sample size.

However, if you create a DataWriter of an unbounded type, it will not communicate with a DataReader of a bounded type out of the box. *RTI DDS Toolkit* sets the following property in all its DomainParticipants:

```xml
<property>
  <value>
    <element>
      <name>dds.type_consistency.ignore_sequence_bounds</name>
      <value>1</value>
    </element>
  </value>
</property>
```

This property allows bounded DataReaders to communicate with unbounded DataWriters. Set this property in your external DDS applications that need to communicate with *RTI DDS Toolkit* applications.

**To Achieve Backward Compatibility:**

If you need to create a bounded string, do not set to true the flag `forceUnboundedString` in the Advanced Reader/Writer Configuration controls. Setting this flag will force all strings to be unbounded.

3.4.3 Preventing ‘Type Code Incorrect’ Error when Working with Arrays

If you are forcing the usage of arrays, you may get an error when reading/writing them. To prevent this error, use sequences instead. Sequences, as well as LabVIEW arrays, can be resized and will not cause this error. Sequences are the default mapping of LabVIEW arrays.
If you must use arrays:

When using an array as the input or output for one of the RTI DDS Toolkit subVIs, you will need to initialize the array to its maximum size. Arrays within clusters must also be initialized to their maximum size. The resize functionality available in LabVIEW is not compatible with RTI DDS Toolkit.

To increase the size of an array, drag down on the bottom of the last element until you’ve reached the largest number of elements you need. Then assign a default value to each new element. It is usually sufficient to add one element at the end of the array.

3.4.4 Troubleshooting with Ping and Spy

If data is not flowing between the writer and reader, we suggest running the Connext DDS Ping and Spy utilities; they can show you what data is flowing through the network. These utilities are provided with the Connext DDS core.

If you do not have Connext DDS installed, you can download RTI Connext DDS Professional from www.rti.com/downloads. Once you’ve installed RTI Connext DDS Professional, you can access DDS Ping and DDS Spy from RTI Launcher (in the Utilities tab).

For help using Ping and Spy, see the Connext DDS API Reference HTML documentation. For 5.1.0 and lower versions, open <Connext DDS core installation directory>/ndds.<version>/ReadMe.html. However if you are using 5.2.0 or a higher version, look for the file <Connext DDS core installation directory>\ReadMe.html. The documentation is also available here: http://community.rti.com/documentation. Choose an API (C, C++, .NET, or Java), then select Modules, Programming Tools.

You can also use RTI Distributed Logger to help debug your applications. Distributed Logger enables applications to publish their log messages to Connext DDS. The log message data can be visualized with RTI Monitor, a separate GUI application that can run on the same host as your application or on a different host. Since the data is provided in a Topic, you can also use DDS Spy or even write your own visualization tool.


---

1. In the <Connext DDS installation directory>/ndds.<version>/scripts (5.1.0 or lower) or <Connext DDS installation directory>/bin (5.2.0 or higher), look for rtiddsping and rtiddsspy.
2. RTI Launcher is a GUI-based tool provided with RTI Connext DDS Professional.
Chapter 4  Tutorial

This tutorial will help you become familiar with several key capabilities of RTI DDS Toolkit. The tutorial assumes you have the following software installed:

- National Instruments LabVIEW 2015 SP1 (32-bit) or later for Windows systems
- RTI DDS Toolkit for National Instruments LabVIEW 2015 SP1 (32-bit) or higher for Windows systems

The tutorial includes these lessons:

- Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) (Section 4.1)
- Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2)
- Lesson 3—Filtering Data (Section 4.3)
- Lesson 4—Reading Only New Samples (Section 4.4)
- Lesson 5—Using Keyed Types (RTI Shapes Demo) (Section 4.5)
- Lesson 6—Used Nested and Multiple Keys (Section 4.6)
- Lesson 7—Reading All Samples (Reliable Communication) (Section 4.7)
- Lesson 8—Debugging Your RTI Connext DDS Application (Section 4.8)
- Lesson 9—Using RTI DDS Toolkit on NI Targets (cRIO-9068 Example) (Section 4.9)
- Lesson 10—Using Security with RTI DDS Toolkit (Section 4.10)

We encourage you to follow along and perform the steps in each lesson yourself—there is no better teacher than hands-on experience. However, completed solutions are provided; see Section 4.11.

Notes:

- These lessons assume you are familiar with LabVIEW.
- For debugging information, see Enabling Debugging Mode (Section E.1)
4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric)

In this first lesson, you will become familiar with the RTI DDS Toolkit functions and capabilities by creating two LabVIEW VIs that can publish and subscribe to data. You can run these VIs on the same computer or separate computers connected to the same local area network. RTI DDS Toolkit will automatically discover the location of each application and handle communication in either scenario without any changes to the VIs.

4.1.1 Developing a VI to Publish Simple Data (Numeric)

Let's start by developing a VI to publish a simple data type: the value of a double-precision numeric control, a LabVIEW Numeric (DBL).

4.1.1.1 Create a Writer Object to Publish a Numeric (DBL)

1. Launch LabVIEW and create a new VI. Select File, New VI. Save the new VI with the name Tutorial_Write_Double.vi.

2. Open the Block Diagram’s Functions Palette (right-click on an open area) and select Data Communication, RTI DDS Toolkit, Writer; drag and drop the Simple Create Writer subVI into the Block Diagram.

3. The Simple Create Writer subVI has the following input parameters:
   - Domain Id
   - Topic Name
   - Data Type
   - error in (no error)

   For details on these parameters, see Writer (Section A.2.1).

   We will use this subVI to create a Writer object that can publish a data type of Numeric (DBL). We will use domain ID 0 and our Topic Name will be Hello LV Double. To begin:

   a. Right-click on the Create Writer subVI and select Select Type, Numeric (DBL)

   b. Right-click on each input node (except error in (no error)) and select Create, Constant. This will create a default constant for that input parameter. Set each input parameter as follows (right-click on each and select Edit...):
      - Domain Id = 0
      - Topic Name = HelloLV Double
      - Data Type = 0

   c. For error in, right-click and select Create, Control.
The resulting Block Diagram should look similar to this:

4.1.1.2 Publish a Numeric (DBL)

The next step is to add the functionality to publish values to the DDS network. We will use the Write subVI.

1. Open the Functions Palette and select Data Communication, RTI DDS Toolkit, Writer, Write; drag and drop the Write subVI into the Block Diagram.

The Write subVI has the following input parameters:
   - DDS Object Ref in
   - Data
   - error in

For details on these parameters, see Writer (Section A.2.1).

2. Wire the DDS Object Ref output of the Create Writer subVI (from Section 4.1.1.1) to the DDS Object Ref in input of the Write subVI.

3. We will publish the value of a Horizontal Pointer Slide control (numeric control). Drop a Horizontal Pointer Slide control onto the Front Panel from the Controls Palette. In the Block Diagram, wire the Pointer Slide to the Write subVI’s Data input node. Rename the slide control to Data.

4. To continuously publish the Pointer Slide value, add a While Loop around the Write subVI in the Block Diagram. From the Functions Palette:
   a. Select Programming, Structures, While Loop.
   b. Use the left mouse button to drag and include both the Write subVI and the Horizontal Pointer Slide control in the While Loop.
   c. You may also add a Wait Until Next ms Multiple subVI (under Programming, Timing from the Functions Palette) inside the While Loop if you want to specify a rate at which Write will publish the value.

5. Add a Stop Button boolean to the Front Panel and wire it to the While Loop stop function in the Block Diagram. The resulting Block Diagram should look similar to this:
4.1.1.3 Release the Writer Object

The final step in our Tutorial_Write_Double.vi is to release the DDS entities and reclaim the system resources when the While Loop is terminated. To do this, we use the Release Writer subVI in the Block Diagram.

1. From the Functions Palette, select Data Communication, RTI DDS Toolkit, Writer, Release Writer; drag and drop the Release Writer subVI into the Block Diagram.
2. Configure its input parameters:
   - DDS Object Ref
   - error in

For details on these parameters, see Writer (Section A.2.1).

Wire the Write subVI’s output to the Release Writer’s inputs. The resulting Block Diagram should look similar to this:

3. Save the file Tutorial_Write_Double.vi.
4. We recommend including error handling in your VIs. Take the above figure as an example: we use error handler’s status to control the loop exit condition.

4.1.2 Creating a VI to Subscribe to Simple Data (Numeric)

In Section 4.1.1, you learned how to develop a LabVIEW VI to use DDS to publish a simple data type, the value of a numeric (DBL). In the second part of the lesson, you will see how to develop an equivalent VI to read the published data.

4.1.2.1 Create a Reader Object to Subscribe to a Numeric (DBL)

1. Launch LabVIEW and create a new VI. (In LabVIEW 2015 SP1, select File, New VI.) Save the new VI with the name Tutorial_Read_Double.vi.
2. Open the Functions Palette (right-click on an open area in the Block Diagram), then select Data Communication, RTI DDS Toolkit, Reader. Drag and drop the Simple Create Reader subVI into the Block Diagram.

3. The Simple Create Reader subVI has the following input parameters:
   - Domain Id
   - Topic Name
   - Data Type
   - error in (no error)

   For details on these parameters, see Reader (Section A.2.2).

   We will use this subVI to create a Reader object that can subscribe to a data type of Numeric (DBL). We will use domain ID 0 and our Topic Name will be Hello LV Double. To begin:
   
   a. Right-click on the Create Reader subVI and select Select Type, Numeric (DBL).
   
   b. Right-click on each input node (except error in (no error)) and select Create, Constant. This will create a default constant for that input parameter. Set each input parameter as follows (by right-click on each and select Edit...):
      - domain id = 0
      - topic name = Hello LV Double
      - data type = 0
   
   c. Right-click on error in and select Create, Control.

   The resulting Block Diagram should look similar to this:
4.1.2.2 Subscribe to a Numeric (DBL)

The next step is to add the functionality to subscribe to the values from the DDS network. We will use the *Read* subVI.

1. To insert the *Read* subVI into your Block Diagram, open the Functions Palette and:
   a. Select *Data Communication, RTI DDS Toolkit, Reader, Read*; drag and drop the *Read* subVI into the Block Diagram.
   b. Right-click on the *Read* subVI and select *Select Type, Numeric (DBL)*.

   *Read* takes the following input parameters:
   - DDS Object Ref in
   - Query Condition
   - Only New Samples
   - error in (no error)

   For details on these parameters, see *Writer (Section A.2.1)*.

2. Wire the *Create Reader* subVI’s *DDS Object Ref* output node to the *Read* subVI’s *DDS Object Ref in* input node.

3. In this example, we will subscribe to the Numeric (DBL) published by the *Tutorial_Write_Double.vi*. To display the data, drop a *Vertical Fill Slide* control onto the Front Panel from the Controls Palette. In the Block Diagram, right-click on the *Vertical Fill Slide* control and select *Change to Indicator*, then wire the *Read* subVI’s *data output* node to the *Vertical Fill Slide*.

4. We want to continuously subscribe to the Numeric (DBL). To do so, add a *While Loop* around *Read* in the Block Diagram. From the Functions Palette:
   a. Select *Programming, Structures, While Loop*.
   b. Use the left mouse button to drag and include both the *Read* subVI and the *Vertical Fill Slide* control in the *While Loop*.
   c. You may also add a *Wait Until Next ms Multiple* function (in the Functions Palette, under *Programming, Timing*) inside the *While Loop* if you want to specify a rate at which *Read* will subscribe to the data.

5. Add a *Stop Button* boolean to the Front Panel and wire the boolean to the *While Loop* stop function in the Block Diagram. The resulting Block Diagram should look similar to this:
4.1.2.3 Release the Reader Object

The final step in our Tutorial_Read_Double.vi is to release the DDS entities and reclaim the system resources when the While Loop execution is terminated. To do this, we use the Release Reader subVI in the Block Diagram.

1. From the Functions Palette, select Data Communication, RTI DDS Toolkit, Reader, Release Reader; drag and drop the Release Reader subVI into the Block Diagram.
2. Configure its input parameters:
   - DDS Object Ref
   - error in

   For details on these parameters, see Reader (Section A.2.2).

   Wire the Read subVI’s outputs to corresponding inputs in the Release Reader subVI.

   The resulting Block Diagram for Tutorial_Read_Double.vi should look similar to this:

3. Save the file Tutorial_Read_Double.vi.
4. We recommend including error handling to your VIs. Please see Section 4.2 and Section 4.3 for further details.

4.1.3 Testing

Now that both VIs are ready, we can verify that they work as expected.

1. Open both VIs, Tutorial_Write_Double.vi and Tutorial_Read_Double.vi, and click the Run arrow button in the toolbar in each.
2. Verify that you are reading exactly the same Numeric (DBL) value in Tutorial_Read_Double.vi that is being published from Tutorial_Write_Double.vi.
While both VIs are running, you can change the value of the Horizontal Fill Slide control in Tutorial_Write_Double.vi and see how the Vertical Fill Slide indicator displays the new values in Tutorial_Read_Double.vi.

These VIs might execute in the same computer or on separate computers connected to the same local area network. Either way, RTI DDS Toolkit will allow the VIs communicate without any changes to the application VIs. This capability is known as ‘location transparency.’

4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters)

In this lesson, you will become familiar with the RTI DDS Toolkit functions and capabilities to publish and subscribe to complex types such as clusters or enumerators.

Note: Only 32-bit enumerators are supported. To change the representation, right-click in the enum and select Representation—>32.

We are going to focus on the cluster use-case. Let’s begin by developing a VI that can publish the cluster defined in Figure 4.1.

First, we will define a new type (a LabVIEW Type-Def) for this cluster:

1. Launch LabVIEW and create a new Custom Control: Select File, New…, Other Files, Custom Control.
2. Choose Type Def. from the Control drop-down list in the toolbar:

3. Draw an empty cluster. From the Controls Palette:
   a. Select Modern
   b. Select Array, Matrix & Cluster
   c. Select Cluster
   d. Rename the cluster complexType (right-click and select Properties).

Note: See Preventing ‘Type Code Incorrect’ Error when Working with Arrays (Section 3.4.3).
4. Fill the complexType cluster as shown in Figure 4.1. This process is simple: drag the following controls from the Palette:

![Palette with controls](image)

- a. String Control labeled as Text.
- b. Numeric Control with Representation I32 labeled as I32_Num (once you have selected a Numeric Control, right-click on it and select Representation and change it to I32).
- c. Numeric Control with Representation I64 labeled as I64_Num.
- d. Numeric Control with Representation U16 labeled as U16_Num.
- e. Array of Numeric Controls with Representation SGL labeled as Sgl_Array.
  
  Note: LabVIEW arrays are mapped as bounded DDS sequences (or arrays if the flag forceArrayMapping is marked in the Advanced Reader/Writer Configuration control). The sequence bound or length is calculated from the LabVIEW array size. Make sure you declare your array to be the maximum size you will need.
- f. Cluster inside the first one labeled as innercluster. Fill innercluster as shown in Figure 4.1.

5. When the cluster definition is complete, save this new control type as Tutorial_Cluster.ctl.

### 4.2.1 Creating VIs for Publishing and Subscribing to a Cluster

In this section, we will demonstrate how to create a set of subVIs to publish and subscribe a cluster using DDS. In order to do that, we will use the DDS ComplexType Generator (see RTI DDS ComplexType Generator (Section 6.3) for further information):

1. Open the RTI DDS ComplexType Generator from the Tools / RTI DDS Toolkit menu.
2. Choose the following information:
   - a. Type of Generation: Advanced.
b. Save the Type Definition: Yes (Note: this may trigger a conflict if your previous type definition has been loaded by LabVIEW).

c. Path to the Custom Type Definition: Path to the Type Definition you want to use for creating this set of subVIs).

Note: If your type contains any arrays, make sure the arrays have been declared with the maximum size you will need. If the array size changes, you will need to regenerate your VIs.

d. Output Directory: The folder where these files will be generated.

e. Generate Example VIs: TRUE. (Note: this will enable the Domain ID and the Topic Name controls).

f. Domain ID: Leave this at 0.

g. Topic Name: The topic name to HelloComplex.

3. Press Generate Code (this button will be enabled when the Path to the Custom Type Definition and the Output Directory controls have values).

4. Press the STOP button.

Several new VIs will be created in the output directory:

- Tutorial_Cluster Create Advanced Reader.vi
- Tutorial_Cluster Create Advanced Writer.vi
- Tutorial_Cluster Read.vi
- Tutorial_Cluster Reader Example.vi
- Tutorial_Cluster Write.vi
- Tutorial_Cluster Writer Example.vi

4.2.1.1 Modify the Writer Example VI

Open Tutorial_Cluster Writer Example.vi and change the advanced setting using the Advanced Writer Configuration control:

1. Disconnect the Advanced Writer Configuration from the Tutorial_Cluster Create Advanced Writer VI.

2. Right-click on Advanced Writer Configuration and select Cluster, Class, & Variant Palette, then Bundle by Name.

3. Create three fields in Bundle by Name and select values typeName, keyName and, optionally, dataWriterQoSProfile.

4. Set typeName to ComplexType and keyName to Text. Optionally, set the QoS Profile to LabVIEWLibrary::DefaultProfile.

Note: For details on Advanced Settings, see Chapter 6: Advanced Concepts and Settings.
4.2.1.2 Modify the Reader Example VI

Open the Tutorial_Cluster Reader Example.vi and change the Advanced Reader Configuration control the same way explained in Modify the Writer Example VI (Section 4.2.1.1).

**Note:** For details on Advanced Settings, see Chapter 6: Advanced Concepts and Settings.

1. Delete the ContentFilteredTopic Info control, since we are not going to use it in this lesson (See Filtering Data Using ContentFilteredTopics (Section 4.3.2)).

2. Optionally:
   - Wire a DDS Sample Info indicator to the Tutorial_Cluster Read subVI.
   - Wire a false boolean constant to the Only New Samples input of the Tutorial_Cluster Read subVI.

The resulting Block Diagram should look similar to this:
3. Save the file `Tutorial_Cluster Reader Example.vi`.

### 4.2.2 Testing

Now that both VIs are ready, you are ready to verify they work as expected.

1. Open `Tutorial_Cluster Reader Example.vi` and `Tutorial_Cluster Writer Example.vi`. Then run each VI.

2. Verify that you can read exactly the same values for each member of the cluster in `Tutorial_Cluster Reader Example.vi`, being published from `Tutorial_Cluster Writer Example.vi`.

   ![Image of VIs running]

   With both VIs running, you can change the value of the published cluster in `Tutorial_Cluster Writer Example.vi` and see the values update.

3. Create a constant in the `Tutorial_Cluster Read` subVI’s input pin named **Only New Samples** and set it to false. Then modify the value for **Text** in the **Writer**. You will see it flicker on the **Reader** side between the previous and current values. This is the expected behavior because **Text** is our cluster’s key. This means that a new instance is created for each **Text** value provided. Even after reading the sample, the received instance is still alive, so it can be reached from the **Reader**. See **Lesson 4—Reading Only New Samples** (Section 4.4) to learn more about this.
4.3 **Lesson 3—Filtering Data**

In this lesson you will learn how a subscriber can filter data available on the DDS network. First we will filter by using a Query Condition. Then, we will use a ContentFilteredTopic.

This lesson assumes you have successfully completed Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2).

### 4.3.1 Filtering Data Using Query Conditions

1. Open Tutorial_Cluster Reader Example from Lesson 2 and save it as a new VI named Tutorial_Filter_Cluster.vi. The Block Diagram should look similar to this:

2. Replace the Read subVI’s query condition input constant with a text control that we can modify while executing the VI. Right-click on the constant wired to the query condition input of the Read subVI.

3. Select Change to Control.
4. Verify that the new **Query condition** text control is available on the Front Panel, as seen in the figure on the right.

5. Save to file **Tutorial_Filt er_Cluster.vi**.

Now we can use filters to specify a **Query condition** at run time and subscribe to only the Topic data we desire. Let’s test how it works:

6. Run **Tutorial_Write_Cluster.vi** to begin publishing the complex data type (cluster).

7. Run **Tutorial_Filter_Read.vi**. As you will see, all the published data is read by the **Tutorial_Read_Cluster.vi**. This is because the **Query condition** text control is blank and no query condition is being applied.

**Note**: DDS is content aware. That is, each Topic and its data type(s) are known by the middleware. This provides robust application support through capabilities such as content filtering, queries, and advanced tooling.

We will now filter data by content; for example, only read those samples where the cluster field is equal to “valid text”:

8. With the VIs running, enter the following filter text in the **Query condition** text control:

   
   "Text = 'valid text'"

   **Note**: See the screenshot below for the exact **Query condition** entry.

9. Change the **Text** data in **Tutorial_Write_Cluster.vi** to “valid text” and modify the value of some of the other types. Verify that you are reading “valid text” and get updated values of the other types in the reader VI.

10. Verify that when you enter any other text in the **Writer VI Text** field, you do not see “valid text” or the updated values of other types in the **Reader VI**.
11. Here are a few other example Query Conditions you can try:

- “I32_Num > 0”
- “innercluster.Boolean = TRUE”
- “innercluster.Boolean = TRUE and Text = ‘valid text'”
- “innercluster.Boolean = TRUE or Text = ‘valid text'”

### 4.3.2 Filtering Data Using ContentFilteredTopics

In this section we will learn how to use ContentFilteredTopics. This allows us to filter the data in the publisher side. See Advanced Filtering of Data—ContentFilteredTopics (Section 6.9) for further details.

This lesson assumes you have successfully completed the previous lesson in Filtering Data Using Query Conditions (Section 4.3.1).

1. Open Tutorial_Read_Cluster.vi from Lesson 3 and save it as a new VI named Tutorial_Content_Filter_Cluster.vi.

   With DDS, you can filter network data by subscribing to only the Topics of interest. Additionally, DDS provides the capability to filter data within a Topic by using a ContentFilteredTopic. A ContentFilteredTopic will not only makes possible to subscribe to topics but also specify that you are only interested in a subset of the Topic’s data.

2. Add a Content Filtered Topic (if it doesn’t exist).

   a. Create a new control in the front panel, right-click, select RTI DDS Toolkit, RTI DDS ContentFilteredTopic Info. Or in the block diagram, right-click on the pin called ContentFilteredTopic Info. Click on Create->Control. This will create a ContentFilteredTopic with the default configuration.

   b. Create your own filter by filling in the ContentFilteredTopic Name and Filter Expression and attaching those parameters to a Bundle by name function, or by filling out the ContentFilteredTopic Info cluster on the front panel.

   ![ContentFilteredTopic Info](image)

   Note: For details on Advanced Settings, see Configuring Advanced Reader Settings (Section 6.5).

   c. Connect it to the Advanced Create Reader subVI.
3. Save to file `Tutorial_Content_Filter_Cluster.vi`. The resulting block should look similar to this:

The `ContentFilteredTopic` has been configured to read only the samples whose `Text = alas`. Let’s test how it works:

4. Run `Tutorial_Writer_Cluster.vi` to begin publishing the complex data type (cluster).
5. Verify that you are receiving the samples whose `Text` value is ‘alas’.
6. Change several parameters of the cluster that is sent in the `Tutorial_writer_cluster.vi`, also modify `Text` to have the value ‘alas2’.
7. Check that no samples are received.
8. Change the `Text` parameter back to ‘alas’.
9. Verify that you are receiving correct samples again and they contain the correct values.

The following picture shows how a Reader Cluster (left) created with a `ContentFilteredTopic` only receives samples that meet the filter condition from the Writer Cluster (right).
10. Here are a few other filter examples you can try:
   - “I32_Num > 0”
   - “innercluster.Boolean = TRUE”
   - “innercluster.Boolean = TRUE and Text = ‘valid text’”
   - “innercluster.Boolean = TRUE or Text = ‘valid text’”

---

4.4 Lesson 4—Reading Only New Samples

In this lesson you will learn how a subscriber can read every received data or only those that have not been read yet. This lesson assumes you have successfully completed Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2).

1. Open Tutorial_Cluster Reader Example.vi from Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2) and save it as a new VI with the name Tutorial_Only_New_Read.vi. The Block Diagram should look similar to this:

   ![Block Diagram](image)

   With DDS, you can select whether you want to subscribe to all the available samples in the Reader queue or just to the new ones. Using the Read subVI’s Only New Samples input, we can modify this behavior. When set to true, only those samples that have not been read before are returned. When set to false, this indicates we want to re-read old samples, even if we read them in the past. This lesson will demonstrate how this feature may affect your system.

2. Replace the Read subVI’s Only New Samples input constant with a boolean control that we can modify while executing the VI. Right-click the constant wired to the Only New Samples input of the Read subVI.
   a. Select Change to Control.
   b. Verify that the new Only New Samples boolean control is available on the Front Panel.
   c. Save to file Tutorial_Only_New_Read.vi.

Now you can specify whether you want to subscribe to new samples or to any available one. Let’s test how it works:

3. If the DDS Sample Info is not visible on the Front Panel, make it visible by right-clicking on it in the Block Diagram and selecting Show indicator.
4. Set Only New Samples to false.

5. Run Tutorial_Only_New_Read.vi and Tutorial_Cluster Writer Example.vi. As you will see, all the published data is read by Tutorial_Only_New_Read.vi.

6. Modify the Text field, which is a key, in the Writer. The values in the Reader will flicker from the new value to the previous one. In fact, in the DDS Sample Info control, you will see that the data that is no longer published has its DDS_SampleStateKind set to DDS_READ_SAMPLE_STATE, while the new one value is set to DDS_NOT_READ_SAMPLE_STATE. Now we are reading any alive sample published by the Writer, even if we had already read it.
7. Change the **Only New Samples** control to **True**. Now we are only reading the latest published value. Take into account that only one data sample is read each time we call the Read subVI (see Lesson 7—Reading All Samples (Reliable Communication) (Section 4.7)).

**Note:** A different approach is to use Exclusive Readers and 'take' to guarantee that the data will only be read once (see Default Configuration: DDS Entities Created by ‘Simple Create’ SubVIs (Section 6.1) and Writing and Reading using Strict Reliability (Section 4.7.2)).

---

### 4.5 Lesson 5—Using Keyed Types (RTI Shapes Demo)

In this lesson, we will explain the value of Keys in our data-type definition and introduce the powerful concept of DDS Topic instances.

We will use RTI Shapes Demo in this lesson. RTI Shapes Demo is a powerful example application to demonstrate the many capabilities of DDS as well as an easy way to quickly communicate with an external DDS application.

Shapes Demo can publish and subscribe to colored, moving shapes (squares, circles, and triangles). It supports a wide range of QoS parameters.

To complete this lesson, you need to install Shapes Demo, which you can download from www.rti.com/downloads. The Shapes Demo User’s Manual is included with the installation.

**Note:** Shapes Demo uses a default domain ID of 0, which is the same domain ID used by the example VIs in this document. If you use a different domain ID for the VIs, you will also need to change the domain ID for Shapes Demo (see the Shapes Demo User’s Manual for instructions).
4.5.1 Working with Shapes Demo

*Shapes Demo* allows you to publish and subscribe different shapes (the DDS Topic for this example). A ‘ShapeType’ data type is defined as a structure with four members:

- color (string) – it will also be used as the Key for the ShapeType
- x (Long, an I32 in LabVIEW)
- y (Long, an I32 in LabVIEW)
- shapesize (Long, an I32 in LabVIEW)

*Shapes Demo* can publish three different Topics of type ShapeType:

- Square
- Circle
- Triangle

4.5.2 Publishing a Shape (Square)

We will use LabVIEW to publish a square in domain 0. Additionally, we will generate two sine functions for the ShapeType X and Y coordinates in order to move the square in a circular or elliptical pattern.

1. Open **RTI Connext DDS Shapes Writer.vi** from the LabVIEW examples **ShapesDemo** directory under **RTI DDS Toolkit**. (Instructions for finding the examples are in Section 1.6.)

2. Open the Block Diagram and note that the VI is creating a Writer object to publish a ShapeType data with Topic Square. The VI uses *Simulate Signal* functions to generate the X and Y coordinates of each square before the square is published.

*Note:* This example uses a string with a max length of 128. To the maximum length, we have included the max size as the text value in the String in the input of the creation subVI.
3. On the Front Panel, you can change these parameters of the Simulate Signal function: `shapesize`, `color`, `Amplitude y`, `Amplitude x`, `Frequency`, `Offset x` and `Offset y`.

4. Launch Shapes Demo and select the Square option under the Subscribe heading. You will see the dialog below. Select OK.

5. Run RTI Connext DDS Shapes Writer.vi and verify that Shapes Demo displays a blue square moving in circles.
6. Use the Front Panel to make changes to the X and Y amplitude and the frequency control. You should see the effects in the Shapes Demo window. The X and Y amplitude control the square’s trajectory, the frequency varies the square’s speed.

7. Change the shape size and color to vary all the parameters. While the size can be any value, we suggest using values between 0 and 100. The color can be: PURPLE, BLUE, RED, GREEN, YELLOW, CYAN, MAGENTA, or ORANGE.

**Note:** When you change the square’s color, you will still see the blue square. This is because we defined Square as the Topic and Color as the Topic Key (instance). Using Keys allows the definition of a single Topic with multiple instances. When you change the color, you are publishing a new instance of the Square Topic of the type ShapeType.

### 4.5.3 Subscribing to Shapes

Instead of using Shapes Demo to subscribe to the published shapes, let’s create our own RTI Connext DDS Shapes Reader in LabVIEW.

1. Open RTI Connext DDS Shapes Reader.vi from the LabVIEW examples ShapesDemo directory under RTI DDS Toolkit.

2. On the Front Panel, you will see two parts:
   - On the left, the VI shows a table, DDS Data, in which the read shapes will be shown. We also see a switch (DDS Stopped). By clicking on that switch, the VI will start reading samples from DDS and add them to the table. In addition, we can see the information of the currently read sample using Sample Info. We can use the Query condition text box on top to filter data, as explained in Lesson 3—Filtering Data (Section 4.3). Finally, we have the Stop button that stops the whole VI.
Lesson 5—Using Keyed Types (RTI Shapes Demo)

- On the right, we have a text box in which we can select one of the shapes using its key, that is, its color. To select the shape, just add the color as shown in the color column in DDS Data. Once selected, the position of the shape will be shown in XY Graph in real time, while its size will be shown in Shape size.

3. Open the Block Diagram and review the three different processes:
   a. Creating the Reader object and reading:
      - A Reader object is created to subscribe to the type ShapeType and the Topic Square, also providing a correct ShapesDemo cluster in the data type pin.
      - Once created, the Reader object reads data from DDS using the Query Condition introduced in the Front Panel.
      - Those data, however, are only read if the DDS Stopped switch is changed to DDS Running by clicking on it (i.e., if it is true).
      - Sample Info is filled with the information of the currently read sample.

   b. Storing data in the table:
      - Each read datum is unbundled to extract the individual components. Each of these components goes in a different column in the DDS Data table.

Note: Due to a known issue in 'Set Cell Value' calls, plot properties cannot be modified at run time. See more details here: http://www.ni.com/product-documentation/52188/en/#407633_by_Date.
• Since each row corresponds to a unique instance, we select the table row using the cluster’s key, i.e., the color.
• When you push the Stop button, the Reader object is released.

Note: This example uses a string with a max length of 128. To the maximum length, we have included the max size as the text value in the String in the input of the creation subVI.

c. Showing selected instance in the XY Graph:

If a color is selected in the text box on the right of the Front Panel, any read sample of that color will appear in the correct X and Y positions in XY Graph. Valid colors are: PURPLE, BLUE, RED, GREEN, YELLOW, CYAN, MAGENTA, and ORANGE.

The size of that shape will be shown in shapesize.
Lesson 6—Used Nested and Multiple Keys

The previous lesson highlighted the value of using keys in your type definitions. Now let’s see how to provide multiple keys for a single data type. This lesson assumes you have successfully completed Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2). You can also use the provided example, RTI Connext DDS Cluster Example Reader/Writer.vi.

4.6.1 Adding Multiple Top-Level Fields as Keys

1. Open Tutorial_Cluster Reader Example.vi from Lesson 2 and save it as a new VI named Tutorial_MultipleKey_Read_Cluster.vi.

   As you can see in the figure to the right, our cluster is quite complex and includes many fields. In Lesson 5—Using Keyed Types (RTI Shapes Demo) (Section 4.5), we marked Text as a key. Depending on the application, we may want to mark other fields as key. Suppose we want I32_Num to be a key too. That will make Text and I32_Num keys.

2. To mark both Text and I32_Num as keys, modify the Key Name string to include both fields, separated by a semicolon (‘;’).

3. Click Run.

   If you use one of the RTI tools such as RTI Admin Console to view the published/subscribed type, you can see that the equivalent IDL for this use case would be:

   ```idl
   struct ultrainnerClusterType{
      sequence<short,2> I16_Array;  //@ID 0
   };
   //@Extensibility EXTENSIBLE_EXTENSIBILITY

   struct superinnerClusterType{
      double Dbl_Num;  //@ID 0
      ultrainnerClusterType ultrainnerCluster;  //@ID 1
   };
   //@Extensibility EXTENSIBLE_EXTENSIBILITY

   struct innerclusterType{
      float Sgl_Num;   //@ID 0
      boolean Boolean; //@ID 1
      superinnerClusterType superinnerCluster;  //@ID 2
   };
   //@Extensibility EXTENSIBLE_EXTENSIBILITY

   struct ComplexType{
      string<1024> Text; //@key  //@ID 0
      long I32_Num;//@key  //@ID 1
```
4-26

4. Repeat this process using Tutorial_Cluster Writer Example.vi, so they can communicate with each other.

4.6.2 Adding Internal Cluster Fields as Keys (Nested Keys)

For a field inside a cluster, use its fully qualified name. This name consists of the cluster name followed by a period (’.’) and then the field name. For instance, to refer to Sgl_Num, use the string innercluster.Sgl_Num. For Dbl_Num, its fully qualified name is innercluster.superinnerCluster.Dbl_Num.

1. Open Tutorial_Cluster Reader Example.vi from Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2) and save it as a new VI named Tutorial_NestedKey_Read_Cluster.vi.

2. Replace the Key Name string with the following value:

   Key Name
   [Text;B2_Num;innercluster.Sgl_Num;innercluster.superinnerCluster.Dbl_Num]

3. Click Run.

If you use one of the RTI tools such as RTI Admin Console to view the published/subscribed type, you can see that the equivalent IDL for this use case would be:

```idl
struct ultrainnerClusterType{
    sequence<short,2> I16_Array; //@ID 0
};//@Extensibility EXTENSIBLE_EXTENSIBILITY

struct superinnerClusterType{
    double Dbl_Num; //@key
    //@ID 0
    ultrainnerClusterType ultrainnerCluster; //@ID 1
};//@Extensibility EXTENSIBLE_EXTENSIBILITY

struct innerclusterType{
    float Sgl_Num; //@key
    //@ID 0
    boolean Boolean; //@ID 1
    superinnerClusterType superinnerCluster; //@key
    //@ID 2
};//@Extensibility EXTENSIBLE_EXTENSIBILITY

struct ComplexType{
    string<1024> Text; //@key
    //@ID 0
    long I32_Num; //@key
    //@ID 1
    long long I64_Num; //@ID 2
```
unsigned short U16_Num; //@ID 3
sequence<float,4> Sgl_Array; //@ID 4
innerclusterType innercluster; //@key
//@ID 5
//@Extensibility EXTENSIBLE_EXTENSIBILITY

Notice that innercluster and superinnercluster are both marked as keys. This is done automatically by RTI DDS Toolkit and is needed for a correct key specification.

Remember that the key name specification is case sensitive.
4.7 Lesson 7—Reading All Samples (Reliable Communication)

This lesson explains how to use LabVIEW to read all the available samples in our Reader. This lesson focuses on sending information reliably. There are two different approaches: using the default RTI DDS Toolkit behavior (see Default Configuration: DDS Entities Created by ‘Simple Create’ SubVIs (Section 6.1)) or using exclusive Reader nodes.

The first approach is explained in Writing and Reading Reliably Using the Default Configuration (Section 4.7.1). The latter approach is explained in Writing and Reading using Strict Reliability (Section 4.7.2).

4.7.1 Writing and Reading Reliably Using the Default Configuration

In our QoS file, there is an already prepared profile to enable this kind of communication: ReliableProfile.

4.7.1.1 Writing Reliably

1. Open a blank VI and open the Block Diagram.

Add an Advanced Create Writer subVI and fill in the parameters to create a Writer object of doubles, as shown in the figure. Pay attention to the new QoS Profile.

For details on the Advanced Create Writer subVI, see Chapter 6: Advanced Concepts and Settings.

2. Create a While Loop and put a Write subVI inside it. We are going to send the loop counter through DDS, so attach that counter to the Writer’s data field. You can also visualize that value by attaching an indicator to the counter. Make sure that the working type of data is DBL, if it is not, the error 5002 can be triggered. In order to modify the data type, right-click on the VI / Select Type / Numeric (DBL). Besides, if you want to delete the coercion point (the red one), you can also add a casting from INT32 to DBL with the function Mathematics / Numeric / Conversion / To Double Precision Float.

3. Add a Release Writer subVI and complete the VI as shown in the following figure. Pay special attention to the Wait function.

4. Save it as Tutorial_Write_Reliable.vi.
4.7.1.2 Reading Reliably

1. Open a blank VI and create an indicator of an array of doubles. Show the vertical scroll bar of the array in the array properties, i.e., right-click in the array, select Properties and check the Show Vertical Scroll Bar option.

2. Add an Advanced Create Reader subVI and fill in the parameters to create a Reader of doubles, as shown in the following figure. Pay close attention to the new QoS Profile.

![Advanced Create Reader configuration](image)

For details on the Advanced Create Reader subVI, see Chapter 6: Advanced Concepts and Settings.

3. Optionally, add a user-defined Invoke note to call the method Reinitialize All to Default. This function resets all the controls and indicators in the VI to the default value. To include it, follow these steps:
   a. Find Invoke Node under Programming, Application Control.
   b. Right-click on the invoke node and go to Select Class, VI Server, VI, VI.
   c. Click on the method label and navigate to Default Values, Reinitialize All to Default.

4. Now we need to read data and discard those values that are not valid. For that:
   a. Add a Read subVI inside a While Loop.
   b. Connect the Read subVI to the Create Reader subVI.
   c. Set Only New Samples to true.
   d. Attach an unbundle function to the DDS Sample Info cluster and select valid_data. This field will be true if the data is a valid one.
   e. If the type of the output data wire is not DBL, you need to modify it manually. To do so, right-click on Read VI, then select Type, Numeric (DBL).

For details on the Read subVI, see Reader (Section A.2.2).
5. If the data is valid, insert it in the array. Otherwise, ignore the data:

![Diagram of Case Structure and While Loop]

a. Create a **Case Structure** from *Programming, Structures* and connect the output of **valid_data** to the question mark.
b. Create an array indicator and connect it to the output of the **Case Structure**.
c. Connect the **Read** subVI outputs as inputs of the **Case Structure**, except **Sample_info** cluster.
d. Create an empty array outside the **While loop** and connect it as input to the **Case Structure**.
e. In the **True** case, add an **Insert into Array** subVI. Connect the empty array and read value inputs as shown above. Connect the output array to the output of the **Case Structure** and to **Array**.
f. In the **False** case, just wire the array input to the output array and to **Array**.
g. Make sure that **ref num out** and **error** wires are also forwarded by connecting them as shown in the image above.

6. Attach the exit of the **Case Structure** to the **While Loop**. Then replace it with a shift register by right-clicking on it and selecting **Replace with Shift Register**. Place the input shift register on the left side of the loop and connect it as an input in the **Case Structure** as shown below.

![Diagram of Case Structure and Shift Register]
7. Add a Release Reader subVI and an Error Dialog. The final Block Diagram should look like the following figure. Pay attention to the reading ratio, it needs to be faster than the writer one or you may need to create a new QoS Configuration File that uses a higher Reader History Depth. See Chapter 5: Loading Quality of Service Profiles for more information.

For details on the Release Reader subVI, see Reader (Section A.2.2).

8. Save it as Tutorial_Read_Reliable.vi.

9. Run the Reader and Writer. You will see how all the data transferred by the Writer arrives at the Reader.

4.7.2 Writing and Reading using Strict Reliability

Writing and Reading Reliably Using the Default Configuration (Section 4.7.1) assumes you are using the default configuration of RTI DDS Toolkit. As explained in Chapter 6: Advanced Concepts and Settings, this configuration uses shared DataReaders, so a more strict reliability (KEEP_ALL History QoS kind and History QoS depth > 1), is not allowed.

If you need strict reliability on your system, you can do it using exclusive readers and the builtin QoS profile: BuiltinQosLibExp::Generic.StrictReliable. This profile is defined internally in RTI Connext DDS (for details on Built-in profiles, see the RTI Community Forum: http://community.rti.com/examples/built-qos-profiles).
4.7.2.1 Writing in Strictly Reliable Mode

1. Open a blank VI and open the Block Diagram.
2. Add a Create Advanced Writer subVI and fill in the parameters to create a Writer object of doubles. Make sure you set the QoS profiles as shown in the following figure:

For details on the Create Advanced Writer subVI, see Chapter 6: Advanced Concepts and Settings.

3. Create a While Loop and put it inside a Write subVI. We are going to send the loop counter through DDS, so attach that counter to the Writer’s data field. You can also visualize that value by attaching an indicator to the counter.

4. Add a Release Writer subVI and complete the VI as shown in the following figure. Pay special attention to the Wait function.

5. Save it as Tutorial_Write_StrictReliable.vi.

4.7.2.2 Reading in Strictly Reliable Mode

1. Open a blank VI and create an indicator of an array of doubles. Show the vertical scroll bar of the array in the array properties, i.e., right-click in the array, select Properties and check the Show Vertical Scroll Bar option.
2. Add an Advanced Create Reader subVI and fill in the parameters to create a Reader of doubles, as shown in the following figure. Make sure you set the QoS profiles and the force-ExclusiveReader? as shown in the following figure.

For details on the Create Advanced Writer subVI, see Chapter 6: Advanced Concepts and Settings.

3. Optionally, add an Invoke note to call the method Reinitialize All to Default. This function resets all the controls and indicators in the VI to the default value. To include it, follow these steps:
   a. Find Reinitialize All to Default under Programming, Application Control.
   b. Right click in the invoke node and go to Select Class, VI Server, VI, VI.
   c. Click in the method label and navigate to Default Values, Reinitialize All to Default.
   d. Connect it as shown in the previous figure.

4. Add a Read subVI inside a While Loop. Connect the Read subVI to the Create Reader subVI. Set Only New Samples to True. Then attach an unbundle function to the DDS Sample Info cluster to check whether the data is valid or not.

For details on the Read subVI, see Reader (Section A.2.2).

5. If the data is valid, insert it in the array. Otherwise, ignore the data:
6. Attach the exit of the If Case to the Loop Case. Then replace it with a shift register by right-clicking on it and selecting Replace with Shift Register. Place the input shift register on the left side of the loop and connect it as an input in the If Case as shown below.

7. Add a Release Reader subVI and an Error Dialog. The final Block Diagram should look like the following figure. Pay attention to the reading ratio: if it is slower than the writer one, you might get an ‘out of resources’ error because the History kind is set to KEEP_ALL.

For details on the Release Reader subVI, see Reader (Section A.2.2).

8. Save it as Tutorial_Read_StrictReliable.vi.

9. Run the Reader and Writer. You will see how all the data transferred by the Writer arrives at the Reader.

4.8 Lesson 8—Debugging Your RTI Connext DDS Application

In this lesson, you will become familiar with the RTI DDS Toolkit QoS profiles and debugging capabilities. RTI DDS Toolkit provides several predefined QoS profiles. You can see the contents of these profiles in the file:

C:/Program Files/National Instruments/LabVIEW 20xx/vi.lib/_RTI DDS Toolkit_internal_deps/RTI_LABVIEW_CONFIG.documentationONLY.xml (where 20xx depends on your LabVIEW version).
In this lesson, we will use two different debugging tools:

- The administration panel to show internal messages about the current execution.
  - Debugging an Application Using the Administration Panel (Section 4.8.1)
- The monitoring profile, which enables RTI Monitoring Library.
  - Adapting a VI to Use RTI Monitoring Library (Section 4.8.2)

### 4.8.1 Debugging an Application Using the Administration Panel

Let’s begin by opening the Reader and Writer VIs creation in Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) (Section 4.1). We are going to get debugging messages from them:

1. Open the Administration Panel. Then in the Tools menu, select RTI DDS Toolkit, RTI DDS Administration Panel. For more details, see Using Administration Panel (for Windows Systems only) (Section 6.7.1).
   
   Note: The Administration Panel may not work on RT Targets. If you want to read messages from a RT Target, you can deploy the VI described in Reading Logged Messages (Section 6.7.2.6).

2. Run the VI.

3. Set the Filter Level to be DEBUG LEVEL. This will cause all messages with log level of Debug or higher to appear in the Debugging table.

4. Press Update to commit the change in the filter level.

5. Now we need to generate some messages. Open the Reader and Writer VIs from Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) (Section 4.1) and click Run.

6. Go back to the Administration Panel. You will see the generated debugging messages in the Debugging table:

---

1. On 64-bit systems, the folder is “Program Files (x86)”
4.8.1.1 Logging Messages Manually

Now that we can debug our application, let’s create our own debugging application. We are going to modify the Writer VI from Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) (Section 4.1) to generate our own logging messages.

1. Save the VI with a different name, such as DebuggingWriter.vi by selecting Save as… in the File menu.
2. Add the Log New Message subVI from the Tools’ Debugging subpalette in the Toolkit palette.
3. Create a Log Level control by right-clicking on the Log Level input in the Log New Message VI. Then choose Create, Control.
4. Add the Format into String function for building a debugging string. Our debugging string will be Published the value $x$, where $x$ is a double number. To do that:
   a. Connect a string constant with the text Published the value at the initial value pin.
   b. Connect a string constant with the text %lf to the format string pin.
   c. Wire the Published Value control to the input 1 pin.
   d. Connect the resulting string to the Message input of the Log New Message subVI, as seen here on the right:
5. Run the Writer VI.
6. Click on the Log Level control and select DEBUG LEVEL.
7. Run the RTI DDS Administration Panel: from the Tools menu, select RTI DDS Toolkit, RTI DDS Administration Panel.
8. Set the Administration Panel’s Filter Level to DEBUG LEVEL as explained in Debugging an Application Using the Administration Panel (Section 4.8.1).
9. Run this new VI and you will see these messages on the administration panel debugging table. The output will be similar to this:

4.8.1.2 Output Provided by RTI Monitor using Distributed Logger

If Distributed Logger is enabled, these messages have been sent through the network and they can be received and shown in RTI Monitor as well.
**Lesson 8—Debugging Your RTI Connext DDS Application**


To send these messages using Distributed Logger and receive them with *RTI Monitor*:

1. Enable Distributed Logger (see Configuration Section (Section 6.7.1.1) for details).
2. Open *RTI Monitor* and join to the domain in which Distributed Logger has been enabled.
3. Select the current process from the list on the left.
4. Create a New Distributed Logger Panel (push this button: ).
5. Use the **State and Controls** tab to set the Filter Level to **Trace**. This allows you to receive all these messages:

![Screenshot of Distributed Logger Panel](image)

4.8.2 **Adapting a VI to Use RTI Monitoring Library**

Let’s begin by opening the **Reader** VI created in Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2): `Tutorial_Read_Cluster.vi`. Or you can use the solution to that lesson mentioned in Section 4.11.

1. Save the VI with a different name, such as `MonitoringReader.vi`, by selecting **Save as...** in the **File** menu.
2. In the Block Diagram, change the qos profile input of the Create Reader subVI to LabVIEWLibrary::MonitoringProfile.

3. Save the VI again.

### 4.8.2.1 Output Provided by RTI Monitor

Now that we have the Monitoring profile loaded in our VI, we can run RTI Monitor to debug our application.


**Important:** Your Path environment variable must include the location of the *RTI Monitoring Library* DLL, *rtimonitoring.dll*, that is noted in Appendix D. Make sure you are not loading *RTI Monitoring Library* from another location.

1. Start *RTI Monitor*; when prompted, join domain 0.

2. Run the original *Tutorial_Read_Cluster.vi*. This example does not enable the monitoring libraries, so *Monitor* will not show useful information. The following snapshot shows the output from *Monitor* when the monitoring libraries are *not* enabled.
3. Stop **Tutorial_Read_Cluster.vi** and make sure that all the entities are released. To do so, close all VIs containing RTI DDS Toolkit subVIs. You can also run the **Release Unused Entities** subVI ten seconds after stopping all the VIs running in the same domain as **Tutorial_Reader_Cluster.vi**.

4. Run **MonitoringReader.vi** and go back to **Monitor**. Now you can see more information such as the topic name, the number of subscribers and publishers, the QoS profile, etc.
4.9 Lesson 9—Using RTI DDS Toolkit on NI Targets (cRIO-9068 Example)

1. Make sure the cRIO is up and running. You can use NI MAX to do so.

2. Follow the installation instructions in Installing RTI DDS Toolkit Support Files on a Target (Section 1.2.1).
3. Create an empty project in LabVIEW by choosing **File, New Project** or **File, Create Project**, depending on your LabVIEW version.

4. Right-click the top-level project item in the Project Explorer window, seen in blue in the above image. Select **New, Targets and Devices** from the shortcut menu to display the Add Targets and Devices dialog box.

5. Select **Existing target or device** and **Specify a target or device by IP address**. Set the correct IP address. Select your device from the list. You can find a list of supported platforms in the 'Supported Platforms' section of the *Release Notes*. Click **OK**.

**Note:** To use the “Discover an existing target(s) or device(s)” option, your host machine must be in the same subnet as your target.
6. Right-click on your new target and select **New, VI**. You can also add an existing one by selecting **Add, File**.

7. Create your application using **RTI DDS Toolkit** as mentioned in the previous lessons. Save it and the project.

8. Once you are finished, run your VI as usual by clicking on the white arrow.
9. LabVIEW will show the Deployment Progress window and will send the VI to your target. This process may take a while, depending on your VI's complexity.

![Deployment Progress Window](image)

Note: If you get an error related to not being able to find rtilvdds.dll, reinstall the RTI DDS Toolkit cRIO support files.

10. Once deployed, you will see a window like this:

![Deployment Progress Window](image)

11. Click Close and work with your VI as you normally would.
4.10 Lesson 10—Using Security with RTI DDS Toolkit

This example is based on the RTI Shapes Demo example in Lesson 5—Using Keyed Types (RTI Shapes Demo) (Section 4.5).

This lesson uses the examples RTI Connext DDS Secure Shapes Reader.vi and RTI Connext DDS Secure Shapes Writer.vi, which are included here: `<LabVIEW installation folder>\examples\RTI DDS Toolkit\SecurityShapesDemo`. This folder also contains a `cert` directory which contains all the necessary files for using DDS Security.

4.10.1 Example Description

This example shows how RTI DDS Toolkit works with RTI DDS Security Plugins. Several scenarios using ShapeType will be shown in this example. Shapes Demo version 5.2.7 or later can be used to graphically show this communication. You can ask for a Shapes Demo trial.

We will use these topics:

- Square
- Circle
- Triangle

And these security profiles:

- AllowAll
- SecureDenySubSquares

These profiles can be created using the instructions in Managing Custom Security Profiles with the Security Panel (Windows Systems) (Section 6.8.1) as well as the provided subVIs described in Managing Custom Security Profiles with SubVIs (Section 6.8.2).

In this example, we will create the security profiles before the creation of the entities that will use them. We will use the created security profile name as the domainParticipantQoSProfile parameter for the Reader/Writer we are creating.

![Diagram of security setup and configuration](image)
We will use the above security profiles to set up a secure environment that uses DDS Security Plugins. You can use this example as a base to create many other security configurations. To do that, you will need to generate your own security certificates, which is explained in the RTI DDS Security Plugins Getting Started Guide, available here: https://community.rti.com/documentation.

The permissions of these profiles will decide how they can behave. The permission file RTI_SHAPES_DEMO_PERMISSIONS.p7s is included in the cert folder. We will use the permissions to create the following security profiles:

- **AllowAll**: This configuration enables secure communication between all the domain IDs and topics. Used by the Writer in the Shapes Security Example. The characteristics of this communication are specified in the Governance. The permissions rule is:

  ```<default>ALLOW</default>```

  **Note:** The security files used by this profile are defined in the file Basic Security Configuration From Path.vi.

- **SecureDenySubSquares**: This configuration won’t allow you to create DataReaders for the topic ‘Square’. The permissions rules are:

  ```
<deny_rule>
  <domain_id>0</domain_id>
  <subscribe>
    <topic>Square*</topic>
  </subscribe>
</deny_rule>
<default>ALLOW</default>
```

  **Note:** The security files used by this profile are defined in the file Create SecureDenySubSquares Profile.vi.

### 4.10.2 Description of VIs

The Secure Shapes Demo example is divided into six VIs:

- **Get full path from file name.vi**: Auxiliary VI which returns a full path that points to the file whose name is **File Name** and is in a subfolder called **cert** in the previous folder level than the current VI.
Basic Security Configuration From Path.vi: Creates a basic security configuration which uses the AllowAll configuration. This subVI will take the security certificates from a sub-folder called cert in the previous folder level than the VI.

Create Security Profile If It Does Not Exist.vi: Creates a Security configuration if the provided name does not exist.

Create SecureDenySubSquares Profile.vi: Adds the files RTI_SHAPES_DEMO_PEER_3 cert and key to the provided base Security Profile. This configuration will use the permissions of SecureDenySubSquares, which denies subscriptions to the topic Square.

RTI Connext DDS Secure Shapes Reader.vi: Main Reader application. This VI will create the DDS entities to subscribe to Shapes. It will be created with the security configuration SecureDenySubSquares.

RTI Connext DDS Secure Shapes Writer.vi: Main Writer application. This VI will create the DDS entities to publish Shapes. It will be created with the security configuration SecureDenyPubCircles.

4.10.3 Main Scenarios

There are two main scenarios (one per topic). All of them occur when we use the specific default Security Profiles for this example. These default Security Profiles are:

Writer: AllowAll
Reader: SecureDenySubSquares

1. Topic Square: If the Writer and Reader are created with the default Security Profiles mentioned above, the Reader won't be able to subscribe to the Square topic. Therefore, no communication will occur.

2. Topics Triangle & Circle: If the Writer and Reader are created with the default Security Profile in the Topic Triangle, the communication will be fine because no restrictions apply to this topic.

4.10.4 Running the LabVIEW Example

Topic Square.

1. Run the Writer using the topic Square and using AllowAll, which is the default profile.
2. The Writer will start to publish a square. You can see it subscribing to the Square topic in Shapes Demo using the AllowAll configuration.
3. Run the Reader with the topic Square using the Security Profile SecureDenySubSquares.
4. You will receive error 5082 because the Security permissions do not allow you to create a DataReader in the topic Square.

Topics Circle & Triangle.

1. Run the Reader using the topics Circle or Triangle with the Security Profile AllowAll or SecureDenySubSquares. By default, the Reader will be created with SecureDenySubSquares.

2. Run the Writer in the topic Circle or Triangle (whichever one you chose before) with the Secure Custom Profile AllowAll, which is the default profile.

No communication will occur because the Writer cannot use the topic Circle, although no error is shown. This happens because the governance property enable_write_access_control is set to false for the Circle topic.

Since there are no restrictions on the Triangle or Circle topics, communication will work fine.

Test your own scenario. For instance:

1. Try to use RTI_SHAPES_DEMO_PEER_2_CERT.pem with RTI_SHAPES_DEMO_PEER_3_KEY.pem in the same Custom Security Profile (no communication will occur).

2. Try to modify these rules (bear in mind that the certificates files may need to be generated again).

3. Try to generate new rules.

4.11 Reviewing Completed Solutions

You can find completed solutions to many of the lessons in this chapter here:

- Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) (Section 4.1)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\NumberDemo

- Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ClusterDemo

- Lesson 3—Filtering Data (Section 4.3)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ClusterDemo
Reviewing Completed Solutions

- Lesson 4—Reading Only New Samples (Section 4.4)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ClusterDemo

- Lesson 5—Using Keyed Types (RTI Shapes Demo) (Section 4.5)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ShapesDemo

- Lesson 7—Reading All Samples (Reliable Communication) (Section 4.7)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ReadAllDemo

- Lesson 8—Debugging Your RTI Connext DDS Application (Section 4.8)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\LogMessagesDemo
  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\MonitoringDemo

- Lesson 9—Using RTI DDS Toolkit on NI Targets (cRIO-9068 Example) (Section 4.9)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\cRIO-9068Project
  (Note: This project is compatible with LabVIEW 2013 and higher)

- Lesson 10—Using Security with RTI DDS Toolkit (Section 4.10)
  \LabVIEW 20xx\examples\RTI DDS Toolkit\SecurityShapesDemo

There is also a GitHub repository with several LabVIEW examples. This repository includes examples that demonstrate single features, as well as real-world examples. The link to the GitHub repository is: https://github.com/rticommunity/rticonnextdds-labview-examples.
Chapter 5  Loading Quality of Service Profiles

This chapter describes how to load personalized QoS profiles in RTI DDS Toolkit.

QoS profiles provide a way to configure your DDS application and define most aspects of the DDS paradigm and the underlying communication mechanisms.

- RTI DDS Toolkit includes a set of predefined QoS profiles. These profiles solve general use-cases such as a Reliable Communication or including RTI Monitoring Library. These profiles are embedded in RTI DDS Toolkit and cannot be modified.

  For your convenience, you can find an XML file that shows you these profiles in C:/Program Files\National Instruments/LabVIEW 20xx/vi.lib/RTI DDS Toolkit/RTI_LABVIEW_CONFIG.documentationONLY.xml (where 20xx depends on your LabVIEW version). As the filename suggests, this file is for documentation purposes only. This file is not loaded by RTI DDS Toolkit, so updating it will not affect the embedded QoS profiles.

- RTI Connext DDS also includes several predefined QoS profiles. You can use these directly from LabVIEW as starting points when creating your own QoS profiles. To access these built-in profiles, use their library name and profile name (for instance, Built-inQosLib::Generic.Monitoring.Common). For more information, consult the RTI Connext DDS Core Libraries User’s Manual (see the chapter on Configuring QoS with XML).

  For information on the format and contents of a QoS profile, consult the RTI Connext DDS Core Libraries User’s Manual (see the chapter on Configuring QoS with XML).

The provided profiles are illustrative and might not fulfill all the desired functionalities. To adjust them to your needs, you can create your own XML configuration file (for instance, USER_QOS_PROFILES.xml). You can define several libraries and profiles in each unique XML file, then refer to their names in subVI calls. For instance, LabVIEWLibrary::DefaultProfile references the DefaultProfile, which you can see in RTI_LABVIEW_CONFIG.documentationONLY.xml.

Once you have defined your desired QoS settings and stored them in a file (or files), RTI DDS Toolkit will load the settings automatically if you point it to the correct file; there are two ways to do this. We strongly recommend the first approach, which provides a more versatile solution.

- Environment variable NDDS_QOS_PROFILES (recommended):

  You can define the environment variable NDDS_QOS_PROFILES and have it point to the XML file that you want to load. You can specify multiple locations for a single XML document via URL groups. The syntax of a URL group is: [URL1 | URL2 | URL2 | ... | URLn].

1. On 64-bit systems, the folder is “Program Files (x86)”
For example:

[file://C:/DDS_config/USER_QOS_PROFILES.xml  
file://C:/DDS_config/ alternative_default_dds.xml]

☐ Working directory (not recommended):

You can save a file called USER_QOS_PROFILES.xml in the working directory of LabVIEW.

The working directory in LabVIEW depends on the application kind. If you are running a VI from LabVIEW, the working directory is the one where the LabVIEW.exe file is, such as C:/Program Files (x86)/National Instruments/LabVIEW 2012/. However, if your application is an independent one, it will use the Run-Time Engine to execute and the working directory will be C:/Program Files (x86)/National Instruments/Shared/LabVIEW Run-Time/2012/.
Chapter 6  Advanced Concepts and Settings

This chapter explains some advanced concepts and describes how to configure advanced parameters in RTI DDS Toolkit.

When configuring an RTI Connext DDS application, there are many parameters that allow you to customize your application. Some of them can be configured by executing using QoS profiles (see Chapter 5: Loading Quality of Service Profiles). Others need to be configured at compile time, such as the topic name and domain ID.

When using RTI DDS Toolkit, you can decide to hide some of that customization to simplify your application, or adapt your settings to match your needs. The first approach uses the Simple Create subVIs. These subVIs only need the mandatory parameters needed for the creation of DataReaders and DataWriters: domain id, topic name and data type.

The second approach uses a more versatile create subVI: Advanced Create Reader/Writer. In the following sections we will explain the different parameters that can be provided to customize your application.

6.1 Default Configuration: DDS Entities Created by ‘Simple Create’ SubVIs

RTI DDS Toolkit has been designed to reduce the number of DDS Entities created and, therefore, minimize the memory and CPU overloads. For example:

- Only one DomainParticipant is created per domain.
- The implicit Publisher and Subscriber are reused, avoiding the creation of new ones.
- Only one DataReader/DataWriter is created per Topic.

When you call the Simple Create subVIs or templates, we internally search for an existing DomainParticipant in the domain, an existing Topic with the correct topic name, and an existing DataReader or DataWriter of the correct data type.

1. When creating complex-type Readers and Writers, you will need to use the Simple Create Reader/Writer subVIs generated by the ComplexType Generator. See Section 6.3 for details.
As an example, consider this scenario.

First we create a Writer VI. Internally, we are creating a DomainParticipant (1), a Topic, and a DataWriter (2). Then, if we create a Reader VI in the same LabVIEW instance, the DomainParticipant and the Topic are reused (3) and only a DataReader is created (4). When a second or third DataReader VIs are created, the DomainParticipant (5), the Topic AND the DataReader are reused (6). This way, all Reader VIs share the same queue.

For most applications, this configuration is sufficient. However, there are several considerations when using shared Entities that may force you to create additional ones:

- If you set the flag ONLY_NEW_SAMPLES to 'true' when reading, only one of the Reader nodes will get the data. This is due to all the Readers sharing the same DataReader.
- Shared DataReaders use 'read' instead of 'take' when getting new data. This prevents shared DataReaders from using the Strict Reliable QoS profile.
- If your application have several Writer nodes for the same Topic, the DataWriter resources need to be adapted to handle the data produced by all the Writer nodes.
- If you need to create DomainParticipants, DataReaders or DataWriters with different QoS properties, you will need to use the Advanced Create subVIs and force the creation of those Entities.
- If you need to set different transport properties, you will need to create different DomainParticipants.

Take into account that having a larger number of DDS Entities requires more resources and will affect performance. So we strongly recommend that you avoid using additional entities whenever possible.

### 6.2 Types with a Specific String Size

Strings in DDS need to know the maximum size at compile time. This maximum size may even be unbounded. However, in LabVIEW, strings dynamically allocate more memory as long as it is needed.
Therefore, when creating a type in DDS which contains a string, we have to somehow provide the maximum length. In LabVIEW, we cannot set the maximum of a string without having any value in it.

To fix this, we can provide the maximum length of the array by typing a number as the value of the string when we create the DataReader/DataWriter. So a string whose text is “75” will be creates as a DDS String of 75 characters. This is the representation of the equivalent IDL:

```cpp
struct MyStruct {
    string myString<75>;
};
```

If the string contains an invalid value (negative number, empty value, non-number, etc.), a string with the default maximum length of 1024 will be created.

This can also be used with a string that is part of a cluster. The only consideration is that the string inside the cluster should contain the value of the maximum size when the DataReader/DataWriter is created.

The figure to the right shows the creation of a DataReader with the type, `ShapeType`. This type will be create with a cluster that contains a string of 128 characters and three 32-bit integers. You can also see the equivalent IDL:

```cpp
struct ShapeType {
    string<128> color; //@key
    long x;
    long y;
    long shapesize;
};
```
6.3 **RTI DDS ComplexType Generator**

RTI DDS ComplexType Generator is a Wizard that allows you to create the basic LabVIEW code needed to run a DDS-based application for Complex Type Definitions. Using the RTI DDS ComplexType Generator is optional if you are using any of the simple types (no clusters).

You can open the ComplexType Generator from the Tools menu (select RTI DDS Toolkit, RTI DDS ComplexType Generator).
Let’s take a look at the ComplexType Generator Wizard:

![RTI DDS Toolkit ComplexType Generator](image)

The ComplexType Wizard asks for several data to generate these LabVIEW subVIs:

- **Type of Generation**: This button allows you to generate the Simple or Advanced creation subVIs (See Configuring Advanced Writer Settings (Section 6.4) and Configuring Advanced Reader Settings (Section 6.5) to learn more about the advanced creation sub-VIs).

- **Save the Type Definition**: Flag to copy the chosen Type Definition to the Output Directory.

- **Path to the Custom Type Definition**: Path to the LabVIEW Type Definition (*.ctl) used for generating the LabVIEW code.

- **Output Directory**: Directory where the generated files are going to be saved.

- **Generation of Example VIs**: Button to enable/disable the generation of a simple DDS-based example using the provided custom Type Definition.

  If a simple example is going to be generated, you can choose the Domain ID and the Topic Name that this example will use.

The **Generate Code** button will be enabled only when the required data is provided (Path to the Custom Type Definition and Output Directory).

The following subVIs will be generated:

- ComplexType Create Simple/Advanced Reader
- ComplexType Create Simple/Advanced Writer
- ComplexType Read
- ComplexType Write
If **Generation of Example VIs** is enabled, the following subVIs will also be generated:

- ComplexType Reader Example
- ComplexType Writer Example

### 6.4 Configuring Advanced Writer Settings

In the Writer subpalette you can find an *Advanced Create Writer.* This subVI is similar to the *Simple Create Writer,* but it has an additional parameter: the *Advanced Writer Configuration* cluster. You can find this cluster in the Control palette: *RTI DDS Toolkit, RTI DDS Advanced Writer Configuration.*

![Advanced Create Writer VI](image)

As you can see in this figure, the cluster allows you to configure the following parameters:

- **typeName**: Name used to register the type in the wire. If this parameter is not provided, a default one is assigned (see default values in *Appendix C*).
- **keyName**: List of fields of a data type that will be marked as key (see Lessons 5 and 6 of the Chapter 4, Tutorial).
- **domainParticipantQoSProfile**: Fully qualified name (Library::Profile) that will be used as a QoS profile when creating the DomainParticipant. If there is an existing DomainParticipant in the same domain ID using a different domainParticipantQoSProfile, a new DDS DomainParticipant will be created using the provided QoS profile. Creating many different DDS DomainParticipants may affect the performance.
- **dataWriterQoSProfile**: Fully qualified name (Library::Profile) that will be used as a QoS profile when creating the DataWriter.
- **forceUnboundedString?**: By default, strings are created with a length of 1024 characters. If this flag is set to true, all strings are created as unbounded (their maximum length corresponds to the maximum 32-bit integer). This configuration optimizes the sample size, sending only the actual data while removing the 1024-character limitation in previous versions of *RTI DDS Toolkit.* This will affect all strings in the data.

---

2. For complex types, use the ComplexTypes Generator in the *Tools/RTI DDS Toolkit* menu. See Section 6.3.
3. Read this article on the creation of multiple DomainParticipants: [https://community.rti.com/best-practices/create-few-domainparticipants-possible](https://community.rti.com/best-practices/create-few-domainparticipants-possible)
6.5 Configuring Advanced Reader Settings

In the Reader subpalette, you can find an `Advanced Create Reader` subVI. This subVI is similar to the `Simple Create Reader`, but it has some additional parameters: the Advanced Reader Configuration cluster and the ContentFilteredTopic Info cluster. You can find these clusters in the Control palette: RTI DDS Toolkit, RTI DDS Advanced Reader Configuration and RTI DDS ContentFilteredTopic Info.

As you can see in the figure, the Advanced Reader Configuration cluster allows you to configure the following parameters:

- **typeName**: The name used to register the type in the wire. If this parameter is not provided, a default one is assigned (see default values in Appendix C).
- **keyName**: List of fields in a data type that will be marked as key (see Lesson 5—Using Keyed Types (RTI Shapes Demo) (Section 4.5) and Lesson 6—Used Nested and Multiple Keys (Section 4.6)).
- **domainParticipantQoSProfile**: The fully qualified name (Library::Profile) that will be used as the QoS profile when creating the DomainParticipant. If there is an existing DomainParticipant with the same domain ID using a different domainParticipantQoSProfiles, a new DDS DomainParticipant will be created using the provided QoS profile. Creating many different DDS DomainParticipants may affect performance.

---

4. For complex types, use the ComplexTypes Generator in the Tools/RTI DDS Toolkit menu. See Section 6.3 for details.
6.6 Working with Instance State Kind

The "DDS Sample Info" structure contains the value of the DDS_InstanceStateKind (aka Instance State). The Instance State is a per-instance concept. An instance is a unique element of a specific DataType within a Topic, described by unique values of key fields.

The values that the Instance State can have are:

- **ALIVE**: The following are all true: (a) DDS samples have been received for the instance, (b) there are live DataWriters writing the instance, and (c) the instance has not been explicitly disposed (or more DDS samples have been received after it was disposed).
- **NOT_ALIVE_DISPOSED**: The instance was explicitly disposed by a DataWriter by means of the dispose() operation.
- **NOT_ALIVE_NO_WRITERS**: The instance has been declared as not-alive by the DataReader because it has determined that there are no live DataWriter entities writing that instance.
6.6.1 Write, Dispose or Unregister

As previously mentioned, you can have three different instance state values. Some of them require a specific action by the user (e.g., call the dispose() function).

A LabVIEW writer will be able to perform different actions. These actions can be chosen by selecting the corresponding value in the ‘Write Sample Kind’ enum input of the Write subVI. The possible options are:

- **WRITE** (Default value): we will write a DDS sample whose data is the information that it has in the input.

- **DISPOSE**: the dispose() function will be called based on the input data. Only the key will be used to dispose a specific instance. This will produce a ‘dummy’ sample whose valid_data is false and the Instance State is NOT_ALIVE_DISPOSED. Also this will change the current Instance State of the samples in the reader queue for the specified instance.

- **UNREGISTER**: this action will unregister an instance. This means that the DataWriter is no longer going to write data for that instance. So, if no other DataWriter is writing data of that instance, the instance state of that instance is NOT_ALIVE_NO_WRITERS. This will produce a ‘dummy’ sample whose valid_data is false and the Instance State is NOT_ALIVE_NO_WRITERS. Also this will change the current Instance State of the samples in the reader queue for the specified instance.

Differences between unregister and dispose:

- When an instance is unregistered, it means this particular DataWriter has no more information/data on this instance.

- When an instance is disposed, it means the instance is “dead”—there will no more information/data from any DataWriter on this instance.

For more information about dispose and unregister, see the *RTI Connext DDS Core Libraries User’s Manual*.

6.6.2 Reading Instance State Kind

Writing instance state samples (or modifying the instance state of a specific instance) doesn’t make sense if you cannot get that information.

The exclusive readers that are not forced to use read() will be able to get samples whose instance state is different that ALIVE. You can only use take() to read NOT_ALIVE instance state samples (either NOT_ALIVE_NO_WRITERS or NOT_ALIVE_DISPOSED). (Otherwise memory leaks could appear, since NOT_ALIVE samples are never taken.)

Once the reader receives a ‘dummy’ sample whose instance state is NOT_ALIVE, it will return the value of the key of that instance.
Note: Since LabVIEW cannot unset some parameters like numbers, it will print the default value, 0 for numbers, empty string for strings. **The user is responsible for making sure to only use the key parameters for that specific DataType.**

The following pictures shows those ‘dummy’ samples:

Using the VIs created in Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) (Section 4.2), we can show this behavior. The Writer sends a `valid_data` whose key ‘Text’ is ‘alas’ (using WRITE in the Write subVI). Then we dispose the instance whose key ‘Text’ value is ‘alas’ (using DISPOSE in the Write subVI). The Reader will look like this:
In the Reader part, we need to force the use of ExclusiveReader to read the NOT_ALIVE instance state samples.

The DDS Sample Info shows the DDS_NOT_ALIVE_DISPOSE_INSTANCE_STATE as the DDS_InstanceStateKind and valid_data is false, because this is the ‘dummy’ sample which indicates the new instance state of the entity.

Finally, we can see how the complexType has returned the key of the disposed instance. In this case, the key is ‘Text’ and the value is ‘alas’. Other values of the cluster are set to the default value. The user is responsible for only reading the key values in case it is necessary.

6.7 Debugging an RTI Connext DDS LabVIEW Application

In the Tools’ DDS Debugging subpalette you can find several subVIs to debug your application. All applications that use RTI DDS Toolkit will create log messages that can be read from the queue in which they are stored. These messages are composed of three parameters:

1. Timestamp, which is the date and time when the message was logged. It is automatically taken from the system clock.
2. Log Level, which is an indicator of the severity of the message. The available levels, from highest severity to lowest are:
   - Fatal
   - Severe
   - Error
   - Warning
   - Notice
   - Info
   - Debug
   - Trace
   - Silent: This level means that the message will never be stored on the queue.
3. Message, which is a string containing useful information.

<table>
<thead>
<tr>
<th>Time</th>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:23:19 PM</td>
<td>DL</td>
<td>This is a Fatal test message.</td>
</tr>
</tbody>
</table>

As mentioned before, all messages are stored in a queue. In addition to the automatically generated messages, you can create and store your own messages (see Logging Messages from LabVIEW (Section 6.7.3)). The queue has associated two configuration parameters:

- Filter Level. Messages with a log level less severe than this Filter Level are not logged. Default value: Warning level.
- Maximum number of elements. If a new message is added to the queue and it is full, the oldest message is deleted. Default value: 512 elements.

Let’s see how the filter level restriction works with an example: the filter level is Warning Level and my application stores the following messages:

- Message 1 with Error level. It is logged.
- Message 2 with Warning level. It is logged.
- Message 3 with Debug level. It is not logged.

*Which kinds of messages can be logged?*

There are three different ways to log new messages into the queue:

- **From the internal RTI Logger.**
  These messages are automatically generated by the internal DDS functionality.

- **From RTI DDS Toolkit.**
  These messages are generated for the LabVIEW integration with DDS.

- **Explicitly from your LabVIEW application.**
  These messages are generated manually using the subVI *Log New Message.vi* (Logging Messages from LabVIEW (Section 6.7.3)).

However, once they are in the queue, all messages are treated equally.

### 6.7.1 Using Administration Panel (for Windows Systems only)

The *RTI DDS Toolkit* Administration Panel is a tool that allows you to administer your DDS applications running on LabVIEW. It also shows diverse DDS information and debugging messages.

The Administration Panel is only supported on Windows systems. This VI uses System Events, which are not supported on Real-Time (RT) targets; therefore the VI is not supported on RT targets. For details on how to debug RT targets, see Debugging SubVIs on Real-Time Targets and Windows Systems (Section 6.7.2).
You can open the Administration Panel from the Tools menu (RTI DDS Toolkit, RTI DDS Administration Panel).

Let’s take a look at the Administration Panel:
The Configuration section allows you to modify the internal behavior of the toolkit and the Administration Panel itself. See Configuration Section (Section 6.7.1.1).

The DDS state cluster shows information about the internal DDS entities created using RTI DDS Toolkit. See DDS State Info (Section 6.7.1.2).

The Debugging table prints the messages stored in the internal logging queue. See Debugging Table (Section 6.7.1.3).

### 6.7.1.1 Configuration Section

This part of the Administration Panel lets you modify different data:

- **Administration panel refresh period:** Refreshing time to update the shown data. Default: 100 ms.
  
  *Note:* The following values will not be updated until you press the **Update** button.

- **Logger Tab Menu:**
  
  - **Local Logger Tab:** All the information about the Local Logger:
    
    - **Max number table rows:** The maximum number of table rows, as well as the maximum queue size. Default: 512 elements. There are different actions depending on the value of this parameter:
      
      - If 0: The internal queue is deleted.
      - If positive and larger than the previous one: Increase the top queue limit.
      - If positive and lower than the previous one: Delete the oldest elements until the size reaches the new maximum size.
    
    - **Is debugging window enabled?:** Allows you to enable/disable the “old” debugging window shown by LabVIEW. Default: disabled.
      
      If you enable the Debugging window, messages will be printed in both the debugging table (an internal queue) and the debugging window.
      
      *Note:* The order in which the messages are presented is not the same in these two windows. In the Debugging window (right), the new messages are printed in order (oldest on top), while in the Debugging table (left), the new messages are printed in reverse order (newest on top), as you can see below:
The Debugging window is a tool for printing text information from a LabVIEW application. On Windows systems, the Debugging windows looks like the above figure. However, on NI™ Linux® systems, setting this boolean parameter to True enables messages to be logged to the console out port.

- **Distributed Logger Tab:** Distributed Logger will be created with the current values of these parameters when you press **Update**. Then the parameters will be grayed out. To modify these values, first you need to disable Distributed Logger (and click **Update**).

- **Distributed Logger DomainParticipantQoSProfile:** The QoS Profile that will be used by the Distributed Logger DomainParticipant. This should follow the next pattern **Library::Profile**. The default QoS profile will be used if the DomainParticipantQoSProfile is empty.

- **Distributed Logger DomainParticipant ID:** The domain ID to be used when creating the next Distributed Logger DomainParticipant. The default is 0.

- **Distributed Logger Queue Size:** The number of messages Distributed Logger will be able to store without dropping any of them. The default is 512 (the same default as **Max number table rows**).

- **Enable Distributed Logger:** Allows you to enable/disable Distributed Logger.

  **Note:** Disabling Distributed Logger will delete all the internal DDS entities that have been created, so it could take a while.

- **Timeout to delete inactive DDS entities:** Delay (in seconds) that internal DDS entities are kept as “active” after releasing them. After this period, the next release call will definitely delete them. If you set it to 0, DDS entities will be deleted as soon as **Release** subVIs are called. Default: 10 seconds.

- **Filter level:** Determines the minimum log-level that messages must have in order to be added to the internal queue. The default value is **WARNING LEVEL**.
6.7.1.2 DDS State Info

This cluster shows the entities created by RTI DDS Tool-Kit, as well as the internal DDS entities:

- **Last number of LabVIEW DDS Nodes**: Number of nodes (Readers and Writers) that were created in the last execution.
- **Current number of LabVIEW DDS Nodes**: Number of nodes (Readers and Writers) that are currently running in the system.
- **Peak number of LabVIEW DDS Nodes**: Maximum number of nodes that has been created in the current execution.
- **Number of DomainParticipants**: Number of DDS DomainParticipants currently active.
- **Number of DataReaders**: Number of active DDS DataReaders.
- **Number of DataWriters**: Number of active DDS DataWriters.
- **Number of Topics**: Number of active DDS Topics.

6.7.1.3 Debugging Table

This table prints the logged messages stored in the internal queue. There are several actions available to manage this table:

- **Clear Table**: Deletes all the printed information.
- **Save as...**: Saves the current state of the debugging table.
- **Clicking on a cell**: Shows the message contained on the pressed cell in the “Full message” box.

6.7.2 Debugging SubVIs on Real-Time Targets and Windows Systems

As mentioned in Section 6.7.1, the Administration Panel is not supported on RT Targets. Instead, you can use the following subVIs to debug and administer RTI Connext DDS applications deployed on RT targets. These subVIs are in the DDS Debugging subpalette under the Tools category. For Windows applications, you can use the Administration Panel, as well as the following subVIs.
6.7.2.1 Get Configuration Parameters

This subVI returns the current configuration parameters explained in Configuration Section (Section 6.7.1.1):

- Timeout to delete inactive DDS entities
- Filter level
- Maximum size of the local queue
- Is debugging window enabled?

These parameters are global to all RTI DDS Toolkit VIs and remain the same as long as rtild-vdds.dll is loaded in memory.

6.7.2.2 Set Configuration Parameters

This subVI updates the configuration parameters explained above. Similarly, as these parameters are global, this modification will affect to all VIs using RTI DDS Toolkit under the same LabVIEW instance.

6.7.2.3 Get DL Configuration Parameters

This subVI returns the current configuration of the Distributed Logger parameters described in Configuration Section (Section 6.7.1.1):

- Whether Distributed Logger is enabled
- Domain ID used to create Distributed Logger
- Distributed Logger Queue Size

This subVI will return the default parameters if Distributed Logger is not created.
6.7.2.4 Configure Distributed Logger

This subVI allows you to configure Distributed Logger. If you enable Distributed Logger, it will use the current parameters to create an instance of Distributed Logger. If you disable it (that is, “Enable Distributed Logger” is False), the instance will be deleted (the other parameters are not used). Only one Distributed Logger instance can be created per instance of the toolkit.

These parameters are used when creating an instance of Distributed Logger:

- **Enable Distributed Logger**: If True, enables Distributed Logger. If False, disables Distributed Logger.
- **Domain Id**: The ID of domain in which an instance of Distributed Logger will be created.
- **Distributed Logger Queue Size**: How many messages can be stored in the Distributed Logger Queue.

**Note**: The Distributed Logger Queue Size shouldn’t be lower than the Local Logger Queue Size, because this could make that several messages logged in the Local Logger won’t be sent through Distributed Logger.

- **DomainParticipant QoSProfile**: The QoS Profile that will be used to create the DomainParticipant. The format of this profile will be “Library::Profile”.

6.7.2.5 DDS State Info

This subVI visualizes the DDS entities created by LabVIEW is controlled by the error wire. The data shown is the same as explained in DDS State Info (Section 6.7.1.2).

6.7.2.6 Reading Logged Messages

This subVI reads the oldest non-printed message from the internal queue and appends it to the beginning of the “Debugging table out”.

There are pins connected to it:

- **Inputs**
• **Debugging table in**: Specifies the debugging table in which to append the new sample if it exists.
• **Clear table?**: Clears the table. Default: disabled.
• **Max number of rows**: Sets a new maximum number of rows in the table. Default: 512 rows.
• **error in (no error)**: error input

**Outputs**

• **Debugging table out**: The “debugging table in” with a new message appended if it existed.
• **Print table?**: Indicates whether a new data was added to the table or the table has been cleared, so the table needs to be printed.
• **error out**: Error standard output.

This subVI is designed to be used within a loop that will periodically read the messages one by one. To get a table updated, the correct use of this subVI is seen the following figure. As you can see, the input of this subVI is a shift register, which allows you to keep the previous printed messages.

Finally, the flag **Print table?** improves the performance by only updating the table control if a new message was read (or if the table has been cleared).

You can find this subVI under [https://github.com/rticommunity/rticonnextdds-labview-examples/tree/master/examples/read_logging_messages](https://github.com/rticommunity/rticonnextdds-labview-examples/tree/master/examples/read_logging_messages).

### 6.7.3 Logging Messages from LabVIEW

As we have seen, there are different ways to log a new message into the internal queue. In the Debugging subpalette you can find **Log New Message.vi**, which allows you to log messages explicitly. This subVI requires the following data:

• **Message**: A string with a meaningful message.
6.8 Enabling Security

To enable security for your DDS application, you need to set several DomainParticipant properties to point to the security files created using OpenSSL®. Then to enable security in RTI DDS Toolkit, you need to create a DomainParticipant QoS profile that includes these properties.

DomainParticipant QoS profiles can be loaded from an XML file, as explained in Chapter 5: Loading Quality of Service Profiles.

**Mandatory Properties:**

At a minimum, your security profile must set these mandatory properties to the corresponding files:

- Identity Certificate Authority (CA): `authentication.ca_file`
- Identity Certificate (Signed by Identity CA): `authentication.certificate_file`
- Private Key: `authentication.private_key_file`
- Permissions Certificate Authority (CA) `access_control.permissions_authority_file`
- Governance Document (Signed by Permissions CA): `access_control.governance_file`
- Permissions Document (Signed by Permissions CA): `access_control.permissions_file`

**Optional Properties:**

- Shared Secret Algorithm: `authentication.shared_secret_algorithm`
- Encryption Algorithm: `cryptography.encryption_algorithm`
- Certificate Revocation List: `authentication.crl_file`

For the full list of available security properties, see the RTI DDS Security Plugins Getting Started Guide, available here: [https://community.rti.com/documentation](https://community.rti.com/documentation).

RTI DDS Toolkit allows you to create the above DomainParticipant QoS profiles, including the security properties. These profiles are referred to as Custom Security Profiles. To create Custom Security Profiles, use the DDS Security Subpalette under the Tools palette:

- Tools palette
  - DDS Security subpalette
    - Create Custom Security Profile.vi
    - Delete Custom Security Profile.vi
    - Get Custom Security Profiles List.vi
• Get Security Profile Values.vi

In addition to these subVIs, there is a Security Panel which allows you to efficiently manage your custom security profiles (for Windows systems only).

Note: Custom security profiles are stored in memory, so they need to be created again every time RTI DDS Toolkit is started (each time you close the toolkit, all the subVIs that use the toolkit are closed). You may want to use Bundle by Name and create your own security configuration cluster from constants that have been saved in your code.

6.8.1 Managing Custom Security Profiles with the Security Panel (Windows Systems)

The Security Panel allows you to manage Custom Security Profiles. From this panel, you can:

- Create a Custom Security Profile
- Delete a Custom Security Profile
- See the current Custom Security Profiles
- Get Security Profile Values.vi

The Security Panel is only supported on Windows systems. This VI uses System Events, which are not supported on Real-Time (RT) targets; therefore the VI is not supported on RT targets. For details on how to create Custom Security Profiles on RT targets, see Managing Custom Security Profiles with SubVIs (Section 6.8.2).
You can open the Security Panel from the Tools menu (RTI DDS Toolkit, RTI DDS Administration Panel).
Let’s take a look at the Security Panel.

- **Name of Base DomainParticipant QoS Profile**: Name of the DomainParticipant QoS profile that will be used as the base profile. The Security Panel will create a profile which includes the QoS settings from the base profile plus any security properties set in the Security Panel. Therefore if a QoS setting is in the profile and set in the Security Panel, the latter will be used. This base profile can be a built-in profile, a profile from the loaded XML configuration (following the pattern Library::Profile), or even a Custom Security Profile (must already be created).

- **Basic Configuration**: Allows you to find the mandatory files needed to load and enable security. This sets the mandatory properties noted in *Enabling Security* (Section 6.8).

- **Advanced Configuration**: Sets the optional properties noted in *Enabling Security* (Section 6.8).
- **Current Profiles**: Shows the current Custom Security Profiles. You can also use this tab to load and delete a profile.

- **Name of DomainParticipant QoS Profile with Security Configuration**: Assigns a name to the new Custom Security Profile. This parameter is mandatory and cannot contain whitespaces. No two profiles can share the same name. This name does not need to follow the pattern Library::Profile.

**Note**: The Basic Configuration must be set in a Secure Custom Profile. However if you load a base profile that contains any of the fields in Enabling Security (Section 6.8), you can avoid filling in the parameters that have already been set. If any of the mandatory fields are not set when the Security Custom Profile is created, error 5077 will be thrown.

### 6.8.1.1 Creating Custom Security Profiles

To create a new Custom Security Profile:

1. *(Optional)* Select a ‘Name of Base DomainParticipant QoS Profile’ to inherit the QoS from. Or leave it blank if you want to load the LabVIEWLibrary::DefaultProfile.
2. Fill in the Basic Configuration settings if these parameters haven’t been inherited.
3. *(Optional)* Fill in the parameters in the Advanced Configuration tab.
4. Set a ‘Name of DomainParticipant QoS Profile with Security Configuration’.
5. Press the **Create New Security Profile** button.

You will see a message indicating that the Custom Security Profile has been correctly created.

### 6.8.1.2 Deleting Custom Security Profiles

To delete a Custom Security Profile:

1. Go to the Current Profiles tab.
2. Select the profile you want to delete.
3. Press the **Delete Selected Profile** button.
6.8.1.3 Load Custom Security Profile Values

To load the values from a Custom Security Profile:

1. Go to the Current Profiles tab.
2. Select the profile you want to load.
3. Press the **Load Profiles Values** button.

The values of this profile will be loaded in the Basic and Advanced Configuration tabs.

6.8.2 Managing Custom Security Profiles with SubVIs

As mentioned in Managing Custom Security Profiles with the Security Panel (Windows Systems) (Section 6.8.1), the Security Panel is not supported on RT targets. Instead, you can use the following subVIs to create your own Custom Security Profiles on these systems.

There is a DDS Security subpalette under the Tools category. For Windows applications, you can use the Administration Panel, as well as the following subVIs.

6.8.2.1 Creating Custom Security Profiles

This subVI creates a new Custom Security Profile with the provided data:

- DomainParticipant Base Profile Name
- Security Settings cluster, including Basic Security Configuration and Advanced Security Configuration
- New Custom Security Profile Name

This also returns the Custom Security Profile Name to be used by other subVIs.

These parameters are the same as those described in Managing Custom Security Profiles with the Security Panel (Windows Systems) (Section 6.8.1).

6.8.2.2 Deleting Custom Security Profiles

This subVI deletes a Custom Security Profile based on the provided name:

- Custom Security Profile Name which will be deleted.
6.8.2.3 Getting Custom Security Profiles List

This subVI returns an array of strings with the current created Custom Security Profiles.

6.8.2.4 Get Security Profiles Values

This subVI returns the Security Settings of a specific profile identified by name. The returned Security Settings will contain all the parameters which this Security Profile uses. This means that even if the Security Profile has been created based on another profile that contained any security properties, the returned Security Settings will contain all the parameters the provided Profile Name uses.

6.8.3 Creating DomainParticipants using a Custom Security Profile

Once the Custom Security Profile has been created (from the Security Panel or the Create Security Profile subVI), you can use it to create a Secure DomainParticipant. To do this, when you are creating a new DomainParticipant, you need to set value of the DomainParticipant QoS Profile to the name of the Custom Security Profile you created in Configuring Advanced Writer Settings (Section 6.4).

If the profile has been created using a Custom Security Profile (without a subVI):

![Diagram of Advanced Reader Configuration cluster to provide the domainParticipantQoSProfile in the advanced settings]
If the profile has been created using a Custom Security Profile with a subVI:

![Advanced Reader Configuration](advanced_reader_configuration.png)

## 6.9 Advanced Filtering of Data—ContentFilteredTopics

A ContentFilteredTopic is a Topic with filtering properties. It makes it possible to subscribe to Topics and at the same time specify that you are only interested in a subset of the Topic’s data. It can also be used to limit the number of data samples a DataReader has to process (and store) and may also reduce the amount of data sent over the network.

A ContentFilteredTopic creates a relationship between a Topic, also called the related Topic, and user-specified filtering properties. The filtering properties consist of an expression used to evaluate a logical expression on the Topic content. The filter expression is similar to the WHERE clause in a SQL expression.

Filtering may be performed on either side of the distributed application. (The DataWriter obtains the filter expression and parameters from the DataReader during discovery.)

When batching is enabled, content filtering is always done on the reader side.

A DataWriter will automatically filter DDS data samples for a DataReader if all of the following are true; otherwise filtering is performed by the DataReader.

1. The DataWriter is filtering for no more than `writer_resource_limits.max_remote_reader_filters` DataReaders at the same time.
2. The DataReader is not subscribing to data using multicast.
3. There are no more than 4 matching DataReaders in the same locator.
4. The DataWriter has infinite liveliness.
5. The DataWriter is not using an Asynchronous Publisher.
6. If you are using a custom filter (not the default one), it must be registered in the Domain-Participant of the DataWriter and the DataReader.
7. The DataWriter is not configured to use batching.

6.9.1 Configuring ContentFilteredTopics

A pin in the Create Advanced Reader subVI allows you to create a ContentFilteredTopic using the specified Topic.

As you can see in this figure, the ContentFilteredTopic allows you to configure the following parameters:

- **Filter Type**: Filter used to create the ContentFilteredTopic. Currently, only DDS_SQLFILTER_NAME is available.
- **ContentFilteredTopic Name**: ID of the ContentFilteredTopic.
- **Filter Expression**: Expression that the ContentFilteredTopic will use to filter data during the exchange between the DataReader and DataWriter. Must be a valid expression for the filter class specified using Filter Type.

**Notes:**

- If the ContentFilteredTopic Name or Filter Expression are empty, the function will not create a ContentFilteredTopic, instead it will use the specified Topic. This will be logged in a debug message.
- Error 5088 will appear if an existing ContentFilteredTopic (attached to a DataReader) is being used with different filter expressions. Two ContentFilteredTopics cannot share the same name if they do not share the same expression. This means that a ContentFilteredTopic's filter expression cannot be modified without changing its name. This implies that no Data Readers are using that ContentFilteredTopic.
Appendix A  VI Descriptions

A.1 Controls Palette Types

In the Front Panel’s Controls Palette, in the Addons section, under RTI DDS Toolkit, you will find the following:

DDS Sample Info: This cluster is returned by the Read subVI and shows information about the current sample. valid_data is 1 if the read data is valid, otherwise it is 0.

- DDS_SampleStateKind: U32 Enum
- DDS_ViewStateKind: U32 Enum
- DDS_InstanceStateKind: U32 Enum
- sec: I32
- nanosec: U32
- valid_data: Boolean

DDS State Info: This cluster contains general statistics from RTI DDS Toolkit. It includes the current number of nodes (both Reader and Writer ones), DomainParticipants, DataReaders, DataWriters, and Topics. It also provides historical information such as the last execution’s nodes.

- Last number of LabVIEW DDS Nodes: I32
- Current number of LabVIEW DDS Nodes: I32
- Peak number of LabVIEW DDS Nodes: I32
- Number of DomainParticipants: I32
- Number of DataReaders: I32
- Number of DataWriters: I32
- Number of Topics: I32

RTI DDS Advanced Reader Configuration: This cluster contains the advanced parameters for the Reader Creation. Use this control with the Create Advanced Reader subVI to provide optional parameters when creating a new Reader.

- typeName: String
- keyName: String
- domainParticipantQoS: String
- dataReaderQosProfile: String
Controls Palette Types

RTI DDS Advanced Writer Configuration: This cluster contains the advanced parameters for the Writer Creation. Use this control with the Create Advanced Writer subVI to provide optional parameters when creating a new Writer.

- typeName: String
- keyName: String
- domainParticipantQoS: String
- dataWriterQosProfile: String
- forceArrayMapping?: Boolean
- forceUnboundedString?: Boolean

RTI DDS Security Settings: This cluster contains the security parameters for enabling DDS Security. This is divided in two internal clusters. The Basic Security Settings includes the mandatory properties that we need to set to enable DDS Security. The Advanced Security Settings just includes some additional configuration parameters.

Basic Security Settings Cluster
- Identify Certificate Authority (CA) File Path
- Identify Certificate (Signed by Identity CA) File Path
- Private Key File Path
- Permissions Certificate Authority (CA) File Path
- Governance Document (Signed by Permissions CA) File Path
- Permissions Document (Signed by Permissions CA) File Path

Advanced Security Settings Cluster
- Shared Secret Algorithm: String
- Encryption Algorithm: String
- Certificate Revocation List: File Path

RTI DDS ContentFilteredTopic Info: This cluster contains the parameters that are necessary to create a ContentFilteredTopic.

- Filter Type: Combo Box String
- ContentFilteredTopic Name: String
- Filter Expression: String

RTI DDS Filter Level: Ring that contains the different debugging levels.

RTI DDS Write Sample Kind: This Enum contains the action to perform when sending samples. The values may be: WRITE, DISPOSE, UNREGISTER. See Write, Dispose or Unregister (Section 6.6.1) for more information.
A.2 Functions Palette

In the Block Diagram’s Functions Palette, in the Data Communication section, under RTI DDS Toolkit, you will find the following:

- Writer (Section A.2.1)
- Reader (Section A.2.2)

A.2.1 Writer

Simple Create Writer: Creates a Writer node able to write data to the DDS network. Use the reference generated by this subVI as input to the Write subVI to send data using DDS. Use the Release Writer subVI to release the allocated memory.

- **Input parameters**
  - Domain Id: ID of the domain the application intends to join
  - Topic Name: Name of Topic for which the application will write data
  - Data Type: Control of the data type to be published
  - error in (no error): LabVIEW Error cluster in (optional)

- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to new Writer object
  - error out: LabVIEW Error cluster out (optional)

Advanced Create Writer: This subVI creates a Writer node able to write data to the DDS network. Introduce advanced configurations by using the control RTI DDS Advanced Writer Configuration.ctl. Use the reference generated by this subVI as input to the Write subVI to send data using DDS. Use the Release Writer subVI to release the allocated memory.

- **Input parameters**
  - Advanced Writer Configuration: Controls of type RTI DDS Advanced Writer Configuration that contains the optional parameters
  - Domain Id: ID of the domain the application intends to join
  - Topic Name: Name of Topic for which the application will write data
  - Data Type: Control of the data type to be published
  - error in (no error): LabVIEW Error cluster in (optional)

- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to new Writer object
  - error out: LabVIEW Error cluster out (optional)

Write: Publishes data into a DDS network. It takes a Writer node (generated by Advanced/Simple Create Writer) as an input parameter. The data type of the data to be written must be the same as the data type attached to the Advanced/Simple Create Writer subVI.
Functions Palette

- **Input parameters**

  DDS Object Ref in  Reference (pointer) to Writer object to be used
  Data Control with the data to be published by DDS. Must be of the same type as specified in the Data Type input for the Advanced/Simple Create Writer.
  Writer Sample Kind Enum for choosing between Write data, Dispose, or Unregister (based on the key).
  error in LabVIEW Error cluster in (optional)

- **Output parameters**

  DDS Object Ref out Reference (pointer) to Writer object used
  error out LabVIEW Error cluster out (optional)

**Release Writer**: Releases the memory allocated for a Writer node and prepares the contained entities to be deleted if nothing else is using them. To force the release of the contained entities, use 'Release Unused Entities' when the defined timeout has been reached after releasing the Writer node.

- **Input parameters**

  DDS Object Ref in  Reference (pointer) to Writer object to be released
  error in LabVIEW Error cluster in (optional)

- **Output parameters**

  error out LabVIEW Error cluster out (optional)

**Set Writer QoS**: Applies a new QoS profile to an existing Writer node. If the current QoS cannot be modified at run time, the Writer node remains unchanged.

- **Input parameters**

  DDS Object Ref in  Reference (pointer) to Writer object whose QoS Profile will be changed
  Qos Profile QoS profile to be applied. The expected value is a string providing the QoS library and profile to be read from the XML file (see Appendix D for details on where this file is located).
  error in LabVIEW Error cluster in (optional)

- **Output parameters**

  DDS Object Ref out Reference (pointer) to Writer object used
  error out LabVIEW Error cluster out (optional)

**A.2.2 Reader**

**Simple Create Reader**: Creates a Reader node that is able to read data from the DDS network. Use the reference generated by this subVI as input to the Read subVI to get data from DDS and store it in the appropriate LabVIEW data. Use the Release Reader subVI to release the allocated memory.
Functions Palette

- **Input parameters**
  - Domain Id: ID of the domain the application intends to join
  - Topic Name: Name of the topic for which the application will read data
  - Data Type: Control of the same data type to be read
  - error in (no error): LabVIEW Error cluster in (optional)

- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to new Reader object
  - error out: LabVIEW Error cluster out (optional)

**Advanced Create Reader**: This subVI creates a Reader node able to read data from the DDS network. Introduce advanced configurations by using the control RTI DDS Advanced Reader Configuration.ctl. Use the reference generated by this subVI as input to the Read subVI to get data from DDS and store it in the appropriate LabVIEW data. Use the Release Reader subVI to release the allocated memory.

- **Input parameters**
  - Advanced Reader Configuration: Control of type RTI DDS Advanced Reader Configuration that contains the optional parameters
  - Domain Id: ID of the domain the application intends to join
  - Topic Name: Name of the topic for which the application will read data
  - Data Type: Control of the same data type to be read
  - error in (no error): LabVIEW Error cluster in (optional)

- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to new Reader object
  - error out: LabVIEW Error cluster out (optional)

**Read**: Gets data from the DDS network. It takes a Reader node (generated by the Advanced/Simple Create Reader subVI) as an input parameter. The data is stored in the appropriate LabVIEW data, which is provided as an output parameter.

- **Input parameters**
  - DDS Object Ref in: Reference (pointer) to Reader object to be used
  - Query Condition: Query expression to use when filtering the read samples; empty means no filtering
  - Only New Samples: Specifies whether to read only the new (unviewed) samples (true) or all the available ones (false)
  - error in (no error): LabVIEW Error cluster in (optional)

- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to Reader object used
  - Data: Indicator that will be filled with the data read from DDS. Must be of the same type as the one specified in the Data Type input of the Advanced/Simple Create Reader subVI
Release Reader: Releases memory allocated for a Reader node and prepares the contained entities to be deleted if nothing else is using them. To force the release of the contained entities, use 'Release Unused Entities' when the defined timeout has been reached after releasing the Reader node.

- **Input parameters**
  - DDS Object Ref in: Reference (pointer) to Reader object to be released
  - Qos Profile: QoS profile to be applied. The expected value is a string providing the QoS library and profile to be read from the XML file (see Appendix D for details on where this file is located).
- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to Reader object used

Set Reader QoS: Applies a new QoS profile to an existing Reader node. If the current QoS cannot be modified at run time, the Reader node remains unchanged.

- **Input parameters**
  - DDS Object Ref in: Reference (pointer) to Reader object whose QoS Profile will be changed
  - Qos Profile: QoS profile to be applied. The expected value is a string providing the QoS library and profile to be read from the XML file (see Appendix D for details on where this file is located).
- **Output parameters**
  - DDS Object Ref out: Reference (pointer) to Reader object used

A.2.3 Tools

**DDS Release Unused Entities**: Releases all the entities generated by the Create Reader/Writer subVIs that are not currently in use. An entity is considered ‘not in use” if no nodes have linked it within the defined timeout period. This is a useful way to resolve some of the errors produced when creating new Reader/Writer nodes.

- **Input parameters**
  - error in: LabVIEW Error cluster in (optional)
- **Output parameters**
  - Error Code: RTI DDS Toolkit Error Code (optional)

**DDS Time to LV Time**: Converts a UNIX timestamp (in seconds) to a LabVIEW Time Stamp.

- **Input parameters**
  - X: DBL
A.2.3.1 DDS Debugging SubPalette

Get configuration parameters: Returns the current values of the configuration parameters of the RTI DDS Toolkit: timeout to release unused DDS entities, filter level, maximum size of the internal queue, and a boolean which indicates whether the debugging window is enabled.

- **Input parameters:**
  - error in: LabVIEW Error cluster in

- **Output parameters:**
  - Timeout to Delete Inactive DDS Entities (s) I32
  - Filter Level I32 Ring
  - Maximum Number of Table Rows U32
  - Is Debugging Window Enabled Boolean
  - error out: LabVIEW Error cluster out

Set configurations parameters: Updates the configuration parameters of the RTI DDS Toolkit: timeout to release unused DDS entities, filter level, maximum size of the internal queue and a boolean to enable/disable the debugging window.

- **Input parameters:**
  - Timeout to Delete Inactive DDS Entities I32 - Default: 10
  - Filter Level I32 Ring - Default: WARNING LEVEL
  - Maximum Number of Table Rows U32 - Default: 512
  - Is Debugging Window Enabled Boolean - Default: False
  - error in: LabVIEW Error cluster in

- **Output parameters:**
  - error out: LabVIEW Error cluster out

Get DL configurations parameters: Returns the current configuration values of the Distributed Logger: a boolean which indicates if Distributed Logger is enabled, the domain ID where the Distributed Logger Domain Participant has been created, and the Distributed Logger Queue Size.

- **Input parameters:**
  - error in: LabVIEW Error cluster in

- **Output parameters:**
  - Is Distributed Logger enabled? Boolean
  - Domain ID U32
  - Distributed Logger Queue Size I32
  - error out: LabVIEW Error cluster out
Configure Distributed Logger: Enables and disables Distributed Logger. When this subVI is enabling Distributed Logger, all the other parameters will be used to create it. These parameters are: enable Distributed Logger, Domain Id, Distributed Logger Queue Size, DomainParticipant QoS Profile.

- **Input parameters:**
  - Enable Distributed Logger: Boolean - Default: False
  - Domain Id: U32 - Default: 0
  - Distributed Logger Queue Size: I32 - Default: 512
  - DomainParticipant Qos Profile: String - Default: empty string

- **Output parameters:**
  - error in
  - LabVIEW Error cluster in
  - error out
  - LabVIEW Error cluster out

Get DDS State: Returns general statistics from RTI DDS Toolkit. This includes the current number of nodes (both Reader and Writer ones), DomainParticipants, DataReaders, DataWriters, and Topics. It also provides historical information such as the last execution’s nodes.

- **Input parameters:**
  - error in
  - LabVIEW Error cluster in

- **Output parameters:**
  - DDS State output
  - DDS State Info Cluster
  - error out
  - LabVIEW Error cluster out

Read One Logged Message: Appends a logging message to the table provided as input. It also allows you to limit the maximum number of table rows; and finally, it returns a flag indicating when the table has been modified, so it could be printed just if it has been modified.

- **Input parameters:**
  - Debugging Table in: 2D String table
  - Clear Table?: Boolean
  - Maximum Number of Rows: U32

- **Output parameters:**
  - Debugging Table out
  - String 2D table
  - Print Table?: Boolean
  - error out
  - LabVIEW Error cluster out

Log New Message: Logs a new message into the internal queue.

- **Input parameters:**
  - Message: String
  - Log level: U32 Ring

- **Output parameters:**
  - error out
  - LabVIEW Error cluster out
A.2.4 DDS Security Subpalette

Create Custom Security Profile: Creates a new Custom Security Profile named **New Custom Security Profile Name** based on the QoS defined in **DomainParticipant Base Profile Name**, including the **Security Settings** configuration.

- **Input parameters:**
  - DomainParticipant Base Profile Name: String (optional)
  - Security Settings: Cluster with the Security Settings (includes Basic and Advanced Security Settings)
  - New Custom Security Profile Name: String

- **Output parameters:**
  - Custom Security Profile Name Output: String

Delete Custom Security Profile: Deletes a previously created Custom Security Profile whose name is 'Custom Security Profile Name'.

- **Input parameters:**
  - Custom Security Profile Name: String

- **Output parameters:**
  - error out: LabVIEW Error cluster out

Get Custom Security Profiles List

- **Input parameters:**
  - error in: LabVIEW Error cluster in

- **Output parameters:**
  - Array of Custom Security Profiles: Array of Strings

Get Security Profile Values: Loads all the security properties that the provided profile has been created with.

- **Input parameters**
  - Profile Name: String

- **Output parameters**
  - Security Settings: Cluster with the Security Settings (includes Basic and Advanced Security Settings)

  error out: LabVIEW Error cluster out
Appendix B  Creation and Release of DDS Entities

The table below explains when RTI DDS Toolkit creates and releases DDS entities.

When an entity is released, RTI DDS Toolkit deletes all ‘unused’ entities in the system. An entity is considered ‘unused’ if no nodes have linked it within the defined timeout period since the last subVI using it was released.

All entities (including the DomainParticipant) are created with the QoS values specified in the QoS Profile input to the Create Writer/Reader functions.

**Note:** You can see when entities are created and released in the Debugging window. See Enabling Debugging Mode (Section E.1).

<table>
<thead>
<tr>
<th>DDS Entity</th>
<th>Is Created When…</th>
<th>Is Released When…</th>
</tr>
</thead>
<tbody>
<tr>
<td>DomainParticipant</td>
<td>The Create Writer/Reader functions are called from LabVIEW and there is not already another valid DomainParticipant. If a DomainParticipant does not exist for that Domain Id and DomainParticipantQos name, a new DomainParticipant is created.</td>
<td>An execution ends and no DDS Reader or Writer objects have used the DomainParticipant within the defined timeout period. The DDS Release Unused Entities function is called from LabVIEW and no DDS Reader or Writer objects are using the DomainParticipant.</td>
</tr>
<tr>
<td>Topic ‘x’</td>
<td>The Create Writer/Reader functions are called from LabVIEW and there is not already another valid Topic.</td>
<td>An execution ends and no DDS Reader or Writer objects are using the Topic. The DDS Release function is called from LabVIEW and no DDS Reader or Writer objects are using the Topic.</td>
</tr>
<tr>
<td>ContentFilteredTopic ‘x’</td>
<td>The Create Advanced Reader VI is called from LabVIEW with a valid ‘ContentFilteredTopic Info’ cluster (all the fields have been set).</td>
<td>An execution ends and no DDS Reader objects are using the ContentFilteredTopic. The DDS Release function is called from LabVIEW and no DDS Reader objects are using the ContentFilteredTopic.</td>
</tr>
<tr>
<td>Subscriber</td>
<td>Never. RTI DDS Toolkit uses an implicit subscriber for each DomainParticipant.</td>
<td>Never.</td>
</tr>
<tr>
<td>Publisher</td>
<td>Never. RTI DDS Toolkit uses an implicit publisher for each DomainParticipant.</td>
<td>Never.</td>
</tr>
<tr>
<td>DDS Entity</td>
<td>Is Created When…</td>
<td>Is Released When…</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DataReader for Topic ‘x’</td>
<td>The Create Reader function is called and there is not already another valid DataReader.</td>
<td>• An execution ends and no DDS Reader objects have used the DataReader within the defined timeout period.</td>
</tr>
<tr>
<td></td>
<td>If the forceExclusiveReader flag is true in the Advanced Create Reader, a new DataReader is created.</td>
<td>• The DDS Release Unused Entities function is called from LabVIEW and no DDS Reader objects are using the DataReader.</td>
</tr>
<tr>
<td>DataWriter for Topic ‘x’</td>
<td>The DDS Create Writer function is called from LabVIEW and there is not already another valid DataWriter.</td>
<td>• An execution ends and no DDS Writer objects have used the DataWriter within the defined timeout period.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The DDS Release Unused Entities function is called from LabVIEW and no DDS Writer objects are using the DataWriter.</td>
</tr>
</tbody>
</table>
Appendix C  Supported Data Types and Corresponding IDL

*RTI DDS Toolkit* supports these simple and complex data types:

- **NUMERIC**
  - INT8
  - INT16
  - INT32
  - INT64
  - FLOAT/SINGLE
  - DOUBLE
  
  a. INT8 and UINT8 are both mapped as octets. We recommend using UINT8, since octets are not signed.

- **BOOLEAN**
- **TEXT (STRING)**
- **ENUM**
  - UINT 32

- **ARRAYS OF TYPE**
  - NUMERIC (INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT, DOUBLE)
  - BOOLEAN
  - ENUM

- **CLUSTER WITH ANY COMBINATION OF:**
  - NUMERIC
  - BOOLEAN
  - TEXT (STRING)
  - ENUM
  - ARRAY
  - CLUSTER

For other DDS applications to communicate with VIs that use *RTI DDS Toolkit*, you need to use compatible data types in both applications.
Simple types have fixed IDLs that are listed in Table C.1.

Clusters use a direct mapping of their configuration into a C struct, see Corresponding IDL for Complex Data Types (Section C.1).

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Sample Entry in IDL</th>
<th>Default TypeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT8</td>
<td>struct Int8Struct{ octet value; }</td>
<td>DDS::Octets</td>
</tr>
<tr>
<td>INT16</td>
<td>struct Int16Struct{ short value; }</td>
<td>DDS_Short</td>
</tr>
<tr>
<td>INT32</td>
<td>struct Int32Struct{ long value; }</td>
<td>DDS_Long</td>
</tr>
<tr>
<td>INT64</td>
<td>struct Int64Struct{ long long value; }</td>
<td>DDS_LongLong</td>
</tr>
<tr>
<td>UINT8</td>
<td>struct UnsignedInt8Struct{ octet value; }</td>
<td>DDS::Octets</td>
</tr>
<tr>
<td>UINT16</td>
<td>struct UnsignedInt16Struct{ unsigned short value; }</td>
<td>DDS_UnsignedShort</td>
</tr>
<tr>
<td>UINT32</td>
<td>struct UnsignedInt32Struct{ unsigned long value; }</td>
<td>DDS_UnsignedLong</td>
</tr>
<tr>
<td>UINT64</td>
<td>struct UnsignedInt64Struct{ unsigned long long value; }</td>
<td>DDS_UnsignedLongLong</td>
</tr>
<tr>
<td>FLOAT</td>
<td>struct FloatStruct{ float value; }</td>
<td>DDS_Float</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>struct DoubleStruct{ double value; }</td>
<td>DDS_Double</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>struct BooleanStruct{ boolean value; }</td>
<td>DDS_Boolean</td>
</tr>
</tbody>
</table>
### Table C.1 Simple Data Types and Corresponding IDL

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Sample Entry in IDL</th>
<th>Default TypeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>Default: struct DDS_String{ string&lt;1024&gt; value; }</td>
<td>DDS::String</td>
</tr>
<tr>
<td></td>
<td>Forcing use of unbounded string: struct DDS_String{ string value; }</td>
<td></td>
</tr>
<tr>
<td>ARRAY of the above types</td>
<td>Default: struct ArrayStruct { sequence&lt;short, nDim&gt; value; }</td>
<td>DDS_Default_TypeName</td>
</tr>
<tr>
<td>(This example uses INT16 and nDim elements.)</td>
<td>Forcing use of array: struct ArrayStruct { short value[nDim]; }</td>
<td></td>
</tr>
</tbody>
</table>

a. If you do not provide a TypeName, a “Default TypeName” is assigned depending on the type. This may cause conflicts if several cluster types are defined in the same DomainParticipant.
C.1 Corresponding IDL for Complex Data Types

C.1.1 Clusters

The IDL representation for a cluster depends on its structure and the type name provided in the Create subVI. If the type name is not provided, we assign DDS_DefaultTypeName as the type name. This may cause conflicts if several cluster-types are defined in the same DomainParticipant.

For example, using the cluster in the figure on the left, assume the type name is MyTypeName. The corresponding IDL would be as follows:

```idl
struct MyTypeName{
    string<1024> Text; //@key
    long I32_Num; //@key
    long long I64_Num;
    unsigned short U16_Num;
    sequence<float,4> Sgl_Array;
    innerclusterType innercluster;
};
struct superinnerClusterType{
    double Dbl_Num;
    ultrainnerClusterType ultrainnerCluster;
};
struct ultrainnerClusterType{
    sequence<short,2> I16_Array;
};
struct innerclusterType{
    float Sgl_Num;
    boolean Boolean;
    superinnerClusterType superinnerCluster;
};
```

Note that inner clusters add “Type” to their name to avoid repeating the same name in both type and member. Also note that all the names of the components are joined by underscores instead of using spaces. This prevents compiling errors in other languages such as C, C++, Java or .Net. Please consider interoperability with these languages and avoid invalid names in the cluster components.

---

1. If `forceUnboundedString?` is set to `true`, IDL correspondence will be `string Text;`. And you will need to run the riddsgen with the option `--unboundedSupport`. 

C-4
C.1.2 Enums

The IDL representation for an enum depends on the elements it is composed of. Remember that only 32-bit enums are supported. Also, if no type name is provided, we use DDS_Enum as type name. This may cause conflicts if different enum-types are defined in the same DomainParticipant.

For example, the enum in the figure on the right would have the following IDL representation for a Type Name "MyType":

```idl
struct EnumStruct{
    MyTypeEnum MyType;
}
enum MyTypeEnum {
    example_value_0 = 0,
    example_value_1 = 1,
    example_value_N = 2
};
```

When the enum is inside a cluster, the representation is slightly different, so several enums can be contained in the same cluster. For the cluster on the right (containing two instances of the enum used in the previous example), the corresponding IDL would be:

```idl
struct MyType{
    MyEnum_1Enum MyEnum_1;
    MyEnum_2Enum MyEnum_2;
};
enum MyEnum_2Enum {
    example_value_0 = 0,
    example_value_1 = 1,
    example_value_N = 2
};
enum MyEnum_1Enum {
    example_value_0 = 0,
    example_value_1 = 1,
    example_value_N = 2
};
```
Appendix D  File Folders Installed within LabVIEW

D.1  File Folders on Windows Systems

RTI DDS Toolkit adds the following files to LabVIEW’s folders.
In the paths shown below, LabVIEW 20xx is:

- C:\Program Files\National Instruments\LabVIEW 20xx
Where xx represents the LabVIEW version number (LabVIEW 2015, etc.)

- DLLs
  - \LabVIEW 20xx\vi.lib\_RTI DDS Toolkit_internal_deps

- Control Types and VIs
  - \LabVIEW 20xx\vi.lib\RTI DDS Toolkit\Types
  - \LabVIEW 20xx\vi.lib\RTI DDS Toolkit\VIs

- QoS Profile (for documentation purposes only)
  - \LabVIEW 20xx\vi.lib\_RTI DDS Toolkit_internal_deps\RTI_LABVIEW_CONFIG.documentationONLY.xml

- Examples
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\ClusterDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\ContentFilteredTopicDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\cRIO-9068Project
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\LogMessagesDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\MonitoringDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\NumberDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\ReadAllDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\SecurityShapesDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\ShapesDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\StringsDemo

1. On 64-bit systems, the folder is “Program Files (x86)”
D.2  File Folders on NI Linux Targets

- Libraries
  - /usr/local/rti/lib
- QoS profile
  - /home/lvuser/rti/RTI_LABVIEW_CONFIG.documentationONLY.xml
Appendix E  Troubleshooting

E.1  Enabling Debugging Mode

To debug your VI, you can use the administration panel or the debugging subpalette, which provides information about several different types. For more information, see Debugging an RTI Connext DDS LabVIEW Application (Section 6.7).

E.2  Error Codes and Possible Solutions

Table E.1 shows error codes and possible solutions.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
<th>Possible Reason(s)</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>Something failed in a previous stage (wired error input)</td>
<td>RTI DDS Toolkit found an error status in the input error cluster. It might be due to an error in the previous stage.</td>
<td></td>
</tr>
<tr>
<td>5002</td>
<td>Error handling the provided LabVIEW Data</td>
<td>Check that the type of all transferred/received data is the same and is similar to the one connected to the data type in the Create Reader/Writer subVIs. LabVIEW data connected to the data type pin in the Create Reader or Create Writer does not correspond with the type sent/received or is missing.</td>
<td></td>
</tr>
<tr>
<td>5003</td>
<td>Unable to delete the contained entities of a participant</td>
<td>It is likely that another application is still using an entity of that Participant. Close all the instances before trying to delete the contained entities. You can also delete the unused contained entities by using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools)</td>
<td></td>
</tr>
<tr>
<td>5004</td>
<td>Unable to delete a participant</td>
<td>It is likely that another application is still using that Participant. Close all the instances before trying to delete it.</td>
<td></td>
</tr>
<tr>
<td>5005</td>
<td>Unable to finalize the DomainParticipantFactory.</td>
<td>It is likely that another application is still using the DomainParticipantFactory. Close all the instances before trying to delete it.</td>
<td></td>
</tr>
<tr>
<td>5006</td>
<td>Bad QoS settings (Library::Profile)</td>
<td>QoS setting format is incorrect or does not match with any of the ones existing in the XML file. Check that format is correct (Library::Profile), the XML file exists, and it contains a correct configuration.</td>
<td></td>
</tr>
</tbody>
</table>
Table E.1  **Error Codes**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
<th>Possible Reason(s)</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>5007</td>
<td>Unable to assert (find or create) a Participant.</td>
<td>Possible error in the QoS configuration. You can also use the default configuration by attaching an empty string as input to the Create Reader/Writer subVI. This may be caused by not having an active network interface in the system. If the monitoring library is being used, it needs to be in the PATH.</td>
<td>Review the QoS profile for the Participant. Modify the QoS profile to work without an active network interface as explained in Running without an Active Network Interface (Section E.3).</td>
</tr>
<tr>
<td>5008</td>
<td>Unable to register the type because there exists another entity with same configuration</td>
<td>This might be caused by an unused entity that has not been released. Close the current VI and release unused entities using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools). Then re-open the current VI.</td>
<td></td>
</tr>
<tr>
<td>5009</td>
<td>Unable to get the Participant QoS for a given profile.</td>
<td>Possible error in the QoS configuration. Check that format is correct (Library::Profile), the XML file exists and it contains that profile. You can also use the default configuration by attaching an empty string as input to the Create Reader/Writer subVI.</td>
<td>Review the QoS profile for the Participant.</td>
</tr>
<tr>
<td>5010</td>
<td>Unable to update the number of applications accessing to the Participant (client count property).</td>
<td>This might cause a memory leak when releasing the participant.</td>
<td></td>
</tr>
<tr>
<td>5011</td>
<td>Unable to set the QoS Properties to the participant.</td>
<td>Check that the QoS configuration provided is correct. You can also use the default configuration by attaching an empty string as input to the Create Reader/Writer subVI.</td>
<td>Review the QoS profile for the Participant.</td>
</tr>
<tr>
<td>5012</td>
<td>Unable to get the description of the topic.</td>
<td>Check that the Reader/Writer was correctly created (no previous errors).</td>
<td></td>
</tr>
<tr>
<td>5013</td>
<td>Type connected to the Read/Write function is incompatible with the current implementation or different than the one in the Create subVI</td>
<td>Check that the correct type is connected to the Create subVI. A correct Type Definition is (Library::Type). String length and array size need to be compatible between the Create and the Read/Write subVIs.</td>
<td>If you recently modified the type, releasing the unused entities or reopening the VI might solve the problem. Remember that LabVIEW arrays of more than one dimension cannot be mapped as sequences.</td>
</tr>
<tr>
<td>5014</td>
<td>Unable to assert (find or create) a Topic.</td>
<td>Check that the QoS profile exists in the XML file and that configuration provided is correct. You can also use the default configuration by attaching an empty string as input to the Create Reader/Writer subVI.</td>
<td>Review the QoS profile for the Topic. Make sure you are selecting the correct settings (Library::Profile).</td>
</tr>
<tr>
<td>5015</td>
<td>Unable to get the implicit publisher.</td>
<td>Implicit publisher is needed to create the Writer. Check that the participant configuration is correct and that there are no previous errors.</td>
<td>Review the QoS profile for the Publisher. Make sure you are selecting the correct settings (Library::Profile). You can also use the default QoS setting by attaching an empty string to the qos profile pin of the Create Writer subVI.</td>
</tr>
</tbody>
</table>
## Table E.1  Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
<th>Possible Reason(s)</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>5016</td>
<td>Unable to get all the DataWriters in the given participant.</td>
<td>It might be due to a memory restriction (not enough memory available to recover the existing DataWriters).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this problem.</td>
<td></td>
</tr>
<tr>
<td>5017</td>
<td>Unable to create the DataWriter.</td>
<td>Check that the QoS configuration provided for the DataWriter is correct.</td>
<td></td>
</tr>
<tr>
<td>5018</td>
<td>Unable to get the QoS Properties from a DataWriter.</td>
<td>Check that Create Writer was successful and that the reference passed to the Write function is the one provided as output from the Create function. It might also be a problem in the QoS setting provided (use default ones as a safest option).</td>
<td></td>
</tr>
<tr>
<td>5019</td>
<td>Unable to set the QoS Properties for a DataWriter.</td>
<td>Check that Create Writer was successful and that the reference passed to the Write/Set_QoS_Setting function is the correct one. It might also be a problem in the QoS setting provided (use default ones as a safest option).</td>
<td></td>
</tr>
<tr>
<td>5020</td>
<td>Unable to update the number of applications using a DataWriter.</td>
<td>This might cause a memory leak when releasing the DataWriter.</td>
<td></td>
</tr>
<tr>
<td>5021</td>
<td>Unable to narrow the Dynamic DataWriter.</td>
<td>Implicit subscriber is needed to create the Reader. Check that the participant configuration is correct and that there are no previous errors.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Review the QoS profile for the Subscriber. Make sure you are selecting the correct settings (Library::Profile). You can also use the default QoS setting by attaching an empty string to the qos profile pin of the Create Reader or Create Writer subVI.</td>
</tr>
<tr>
<td>5022</td>
<td>Unable to get the implicit subscriber.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5023</td>
<td>Unable to get all the DataWriters in the given participant.</td>
<td>It might be due to a memory restriction (not enough memory available to recover the existing DataWriters).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5024</td>
<td>Unable to create the DataReader.</td>
<td>Check that the QoS configuration provided for the DataReader is correct.</td>
<td>Review the QoS profile for the DataReader. Make sure you are selecting the correct settings (Library::Profile). You can also use the default QoS setting by attaching an empty string to the qos profile pin of the Create Reader or Create Writer subVI.</td>
</tr>
<tr>
<td>5025</td>
<td>Unable to get the QoS Properties from a DataReader.</td>
<td>Check that Create Reader was successful and that the reference passed to the Read function is the correct one. It might also be a problem in the QoS setting provided (use default ones as a safest option).</td>
<td></td>
</tr>
<tr>
<td>5026</td>
<td>Unable to set the QoS Properties for a DataReader.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td>Possible Reason(s)</td>
<td>Additional Information</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5027</td>
<td>Unable to update the number of applications using a DataReader.</td>
<td>This might cause a memory leak when releasing the DataReader.</td>
<td></td>
</tr>
<tr>
<td>5028</td>
<td>Unable to narrow the Dynamic DataWriter.</td>
<td>This is an unexpected error. Contact <a href="mailto:lab-view@rti.com">lab-view@rti.com</a> or visit our Community Portal at <a href="http://community.rti.com">http://community.rti.com</a> to view current solutions and forum entries.</td>
<td></td>
</tr>
<tr>
<td>5029</td>
<td>Unable to delete a Topic.</td>
<td>It is likely that another instance of LabVIEW is still using that Topic. Close all LabVIEW instances before trying to delete it.</td>
<td>You can also delete the unused contained entities by using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools).</td>
</tr>
<tr>
<td>5030</td>
<td>Unable to delete a DataReader (or its contained entities).</td>
<td>It is likely that another instance of LabVIEW is still using that DataReader or its entities. Close all LabVIEW instances before trying to delete it.</td>
<td></td>
</tr>
<tr>
<td>5031</td>
<td>Unable to delete a DataWriter (or its contained entities).</td>
<td>It is likely that another instance of LabVIEW is still using that DataWriter or its entities. Close all LabVIEW instances before trying to delete it.</td>
<td></td>
</tr>
<tr>
<td>5032</td>
<td>Unable to initialize the DDS Dynamic Data.</td>
<td>There was a problem when allocating memory. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5033</td>
<td>Unable to initialize the Reader Node.</td>
<td>There was a problem when allocating memory. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5034</td>
<td>Unable to initialize the DDS Manager.</td>
<td>Check that the DLL was correctly loaded (a message can be found in the Debug Window).</td>
<td></td>
</tr>
<tr>
<td>5035</td>
<td>Invalid reference to a Reader or Writer Node.</td>
<td>Please use the appropriate Create subVI to generate a correct reference and connect it to the Read/Write subVI.</td>
<td>Pay special attention to the data type.</td>
</tr>
<tr>
<td>5036</td>
<td>Unable to read data from DataReader.</td>
<td>Check that the Query Condition is correctly set.</td>
<td>* will return everything. A regular expression will also work (for instance: Text='hello').</td>
</tr>
<tr>
<td>5037</td>
<td>Unable to initialize the Writer Node.</td>
<td>There was a problem when allocating memory. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5038</td>
<td>Unable to write data.</td>
<td>DataWriter timed out or ran out of resources. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td>Check that you attached a valid indicator/storage to the write output.</td>
</tr>
<tr>
<td>5039</td>
<td>Unable to initialize the semaphore for the DLL.</td>
<td>There was a problem when allocating memory. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td>Possible Reason(s)</td>
<td>Additional Information</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5040</td>
<td>Unable to create the Query Condition to filter Read subVI.</td>
<td>Check that the Query Condition is correctly set. To read everything, set it to * or leave it empty.</td>
<td>A regular expression will also work (for instance: Text=’hello’).</td>
</tr>
<tr>
<td>5041</td>
<td>The type connected to the Create subVI is not supported in the current version.</td>
<td>The Getting Started Guide provides more information about the supported types.</td>
<td>See Appendix C: Supported Data Types and Corresponding IDL.</td>
</tr>
<tr>
<td>5042</td>
<td>Unable to unregister the Type Code.</td>
<td>Other applications might be using it. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5043</td>
<td>The LabVIEW Data Type connected has changed.</td>
<td>The LabVIEW Type changed but wasn’t correctly initialized (using Create subVI). You might need to close the VI and re-open it to removed unused entities.</td>
<td>This error happens if you created and run the Reader/Writer and then you modified the type connected to the create subVI. Close and re-open the VI or use the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools).</td>
</tr>
<tr>
<td>5044</td>
<td>Unable to get all the available Topics.</td>
<td>It might be due to a memory restriction (not enough memory available to recover the existing Topics).</td>
<td></td>
</tr>
<tr>
<td>5044</td>
<td></td>
<td>Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5045</td>
<td>Warning: Unable to delete one or several DDS Entities.</td>
<td>Other instances of LabVIEW are currently using one or several of the DDS Entities.</td>
<td>This is not an error, just a warning. Closing all running VIs should release all the remaining DDS Entities.</td>
</tr>
<tr>
<td>5046</td>
<td>Unable to get the Topic’s QoS.</td>
<td></td>
<td>Review the QoS profile for the Topic. Make sure you are selecting the correct settings (Library::Profile). You can also use the default QoS setting by attaching an empty string to the qos profile pin of the Create Reader or Create Writer subVI.</td>
</tr>
<tr>
<td>5047</td>
<td>Unable to set the Topic’s QoS.</td>
<td>Check that the Topic’s QoS provided was correct and that the Topic was initialized using the Create Reader or Create Writer subVI.</td>
<td></td>
</tr>
<tr>
<td>5048</td>
<td>Unable to access library handler.</td>
<td>Possible error in the QoS properties provided. The RTI DDS Toolkit Dynamic Library was not correctly loaded.</td>
<td></td>
</tr>
<tr>
<td>5049</td>
<td>Unable to take the semaphore</td>
<td>Another thread may already be using the DLL.</td>
<td></td>
</tr>
<tr>
<td>5050</td>
<td>Unable to recover participant’s default QoS</td>
<td>Internal error due to default configuration issues. Contact <a href="mailto:labview@rti.com">labview@rti.com</a> or visit our Community Portal at <a href="http://community.rti.com">http://community.rti.com</a> to view current solutions and forum entries.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td>Possible Reason(s)</td>
<td>Additional Information</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5051</td>
<td>Unable to load QoS profiles from the embedded configuration or external XML files</td>
<td>Error in QoS properties. Verify all profiles loaded by the NDDS_QOS_PROFILES environment variable.</td>
<td>Make sure you are selecting the correct settings (Library::Profile). You can also use the default QoS setting by attaching an empty string to the qos profile pin of the Create Reader or Create Writer subVI.</td>
</tr>
<tr>
<td>5052</td>
<td>Incorrect type name.</td>
<td>Usual format is Library::Type. Avoid using spaces.</td>
<td></td>
</tr>
<tr>
<td>5053</td>
<td>One of the required parameters of the subVI is missing</td>
<td>Required parameters for Create subVIs: domain_id, topic_name, type_name, data_type; for Read/Write subVIs: ref_in and data; for Release: ref_in.</td>
<td>These pins are also required for the clusters even if you use Call Library Function (CLF) calls instead of a subVI.</td>
</tr>
<tr>
<td>5054</td>
<td>Unable to access to the Type Code Factory.</td>
<td>Another application has finalized the Type Code Factory and there was an error while reinitializing it. Retry.</td>
<td></td>
</tr>
<tr>
<td>5055</td>
<td>Unable to add a new member to the Type Code.</td>
<td>The cluster used is incompatible. Make sure all field labels exist and are compatible with text-based languages: no spaces. Make sure all used types are supported.</td>
<td></td>
</tr>
<tr>
<td>5056</td>
<td>Unable to create the Type Code.</td>
<td>The attached cluster is incompatible with the supported one and cannot be created.</td>
<td></td>
</tr>
<tr>
<td>5057</td>
<td>Unable to set the Dynamic Data.</td>
<td>Check that the correct data type is connected to the subVI (pay special attention to Create Reader/Writer ones).</td>
<td></td>
</tr>
<tr>
<td>5058</td>
<td>Unable to get the Dynamic Data.</td>
<td>Check that the correct data type is connected to the subVI (pay special attention to Read/Write ones).</td>
<td>Review the QoS profile for the Reader/Writer. Some QoS setting cannot be applied once the Reader/Writer is created unless you completely delete it. Close and reopen the VI or use the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools).</td>
</tr>
<tr>
<td>5050</td>
<td>Invalid profile provided to the Set QoS subVI.</td>
<td>There may be an incompatible QoS Policy. Check that the provided profile exists. Once created, some QoS settings cannot be modified. Try using that QoS Policy in the Create subVI.</td>
<td></td>
</tr>
<tr>
<td>5060</td>
<td>Unable to give the semaphore.</td>
<td></td>
<td>This might block another thread from using the RTI DDS Toolkit API.</td>
</tr>
<tr>
<td>5061</td>
<td>Unable to lock/unlock the Participant to create the Reader.</td>
<td>Another application was already deleting the Participant. Removing unused entities or closing the VIs might fix this problem.</td>
<td>You can also delete the unused contained entities by using the DDS Release Unused Entities subVI (in RTI DDS Toolkit Tools).</td>
</tr>
<tr>
<td>5062</td>
<td>Reached the maximum number of participants allowed in the system.</td>
<td>Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5063</td>
<td>Unable to create the system clock.</td>
<td>This is an unexpected error. Contact <a href="mailto:labview@rti.com">labview@rti.com</a> or visit our Community Portal at <a href="http://community.rti.com">http://community.rti.com</a> to view current solutions and forum entries.</td>
<td></td>
</tr>
</tbody>
</table>
Table E.1  Error Codes

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
<th>Possible Reason(s)</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>5064</td>
<td>Unable to create the Type Support needed to register a type.</td>
<td>There was a problem when allocating memory. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5065</td>
<td>Unable to assign that type name to the Topic because it is currently in use.</td>
<td>The type name provided is already registered and used by some entities. Using the DDS Release Unused Entities subVI (in RTI DDS Toolkit, Tools) might fix this.</td>
<td></td>
</tr>
<tr>
<td>5066</td>
<td>The attached enum is not a 32-bit one. Only 32-bit enumerators are supported.</td>
<td>The current implementation only supports 32-bit enumerators. Change the enumerator representation or use an 8-bit or 16-bit integer. To change the representation, right-click in the indicator/control and select Representation—&gt;U32.</td>
<td></td>
</tr>
<tr>
<td>5067</td>
<td>Unable to create the key with the provided string. Might be a memory allocation problem.</td>
<td>KeyName should be a string containing the key names separated by semicolons (';'). The fields inside a cluster can be provided in the form 'cluster.field'. See Section 4.6 for further details.</td>
<td></td>
</tr>
<tr>
<td>5068</td>
<td>Unable to create DataReader using read() with KEEP_ALL history kind. Use case not supported.</td>
<td>Use the shipped profile 'LabVIEWLibrary::ReliableProfile' to use Reliable Communication with Shared Readers. If you need Strict Reliability or History kind KEEP_ALL, use Exclusive Readers and do not force them to use read(). The current implementation of a non-exclusive Reader uses 'read' instead of 'take', so strictly reliable communication is not compatible with non-exclusive Readers.</td>
<td></td>
</tr>
<tr>
<td>5069</td>
<td>Incompatible configuration: History depth &gt; 1 needs 'only_new_samples' flag in the Read subVI to be 'true'.</td>
<td>Using a depth bigger than 1 for the history property and not setting the only_new_samples' could cause that samples stayed unread. Change the QoS configuration or set the flag to 'true'. Review the QoS profile for the DataReader.</td>
<td></td>
</tr>
<tr>
<td>5070</td>
<td>Unable to extract information from the Advanced Writer Configuration control.</td>
<td>Make sure you are using the cluster 'RTI DDS Advanced Writer Configuration.ctl' contained in LVDDS_Library.</td>
<td></td>
</tr>
<tr>
<td>5071</td>
<td>Unable to extract information from the Advanced Reader Configuration control.</td>
<td>Make sure you are using the cluster 'RTI DDS Advanced Reader Configuration.ctl' contained in LVDDS_Library.</td>
<td></td>
</tr>
<tr>
<td>5072</td>
<td>The Local Logger is not correctly initialized.</td>
<td>Make sure the size of the Local Logger is not a negative number.</td>
<td></td>
</tr>
<tr>
<td>5073</td>
<td>Unable to create a new message into the Local Logger.</td>
<td>Make sure there is enough memory to log a new message. You could need to use a lower queue size.</td>
<td></td>
</tr>
<tr>
<td>5074</td>
<td>Unable to create Distributed Logger.</td>
<td>Check that the Distributed Logger Queue Size is a positive number and the QoS setting format is correct (Library::Profile), the XML file exists, and it contains a correct configuration.</td>
<td></td>
</tr>
<tr>
<td>5075</td>
<td>Unable to delete Distributed Logger.</td>
<td>Make sure Distributed Logger has not been previously deleted.</td>
<td></td>
</tr>
<tr>
<td>5076</td>
<td>Unable to create the custom QoS list. Might be a memory allocation problem.</td>
<td>The Custom QoS Security Profiles list has not been able to be allocated. Free memory and try again.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td>Possible Reason(s)</td>
<td>Additional Information</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5077</td>
<td>Unable to create the custom QoS Profile. The name might be in use or the input parameters are not correctly set</td>
<td>The Custom QoS Profile cannot be created. Make sure the provided name is not already in use, doesn't contain whitespaces and the input parameters are correctly set.</td>
<td>The new Custom Security Profile provided name cannot be used 2 times. also, make sure that all the Basic Security Configuration parameters have been correctly set.</td>
</tr>
<tr>
<td>5078</td>
<td>Unable to delete the custom QoS Profile. The custom profile might not exist.</td>
<td>The Custom QoS Profile cannot be deleted. Make sure the provided name is created and doesn't contain whitespaces.</td>
<td>The Custom Security Profile you want to delete contains whitespaces or it hasn't been created yet.</td>
</tr>
<tr>
<td>5079</td>
<td>Unable to allocate memory for showing the created custom QoS profiles.</td>
<td>Unable to allocate memory for showing the created custom QoS profiles.</td>
<td></td>
</tr>
<tr>
<td>5080</td>
<td>Unable to assert (find or create) a Secure Participant.</td>
<td>Unable to assert a Secure Participant. Make sure the provided domainID is allowed by the Security Permissions, OpenSSL is in your PATH and the nddssecurity library is in the toolkit installation path.</td>
<td></td>
</tr>
<tr>
<td>5081</td>
<td>Unable to assert (find or create) a Secure Topic.</td>
<td>Check that the QoS profile exists and the TopicName is allowed by the Security Permissions.</td>
<td></td>
</tr>
<tr>
<td>5082</td>
<td>Unable to assert (find or create) a Secure DataReader.</td>
<td>Unable to assert a Secure DataReader. Check that the QoS configuration provided for the DataReader is correct and make sure that the provided DataReader is allowed in that Topic and domainID by the Security Permissions.</td>
<td></td>
</tr>
<tr>
<td>5083</td>
<td>Unable to assert (find or create) a Secure DataWriter.</td>
<td>Unable to assert a Secure DataWriter. Check that the QoS configuration provided for the DataWriter is correct and make sure that the provided DataWriter is allowed in that Topic and domainID by the Security Permissions.</td>
<td></td>
</tr>
<tr>
<td>5084</td>
<td>Unable to load the custom QoS Profile. The custom profile might not exist.</td>
<td>The Custom QoS Profile cannot be loaded. Make sure the provided name exists and does not contain whitespaces.</td>
<td>The Custom Security Profile Name already exists or contains whitespaces.</td>
</tr>
<tr>
<td>5085</td>
<td>Unable to assert (find or create) a ContentFilteredTopic</td>
<td>Unable to assert a ContentFilteredTopic. Check that there is not a different ContentFilteredTopic or any Topic with the same name. Also check that the Filter is supported.</td>
<td>A ContentFilteredTopic cannot be created with the same name that a Topic is using. It can neither share the name with other ContentFilteredTopics with different expression. Currently only the filter type “DDS_SQLFILTER_NAME” is supported.</td>
</tr>
<tr>
<td>5086</td>
<td>Unable to delete ContentFilteredTopic.</td>
<td>Unable to delete a ContentFilteredTopic. Check that no DataReaders are using it.</td>
<td>The ContentFilteredTopic may not exist or any DataReader is still using it.</td>
</tr>
</tbody>
</table>
To use RTI DDS Toolkit on a computer that does not have an active network interface, you have two choices:

- Change the QoS profile to use only the Shared Memory transport. As described in the RTI Connext DDS Core Libraries User’s Manual (see the chapter on Configuring QoS with XML), you need to set up this QoS properties in all your profiles:

  ```xml
  <participant_qos>
    <transport_builtin>
      <mask>SHMEM</mask>
    </transport_builtin>
    <discovery>
      <initial_peers>
        <element>builtin.shmem://</element>
      </initial_peers>
    </discovery>
  </participant_qos>
  ```

- Another option is to install the Microsoft Loopback Adapter, which simulates the existence of a network interface. For an example on how to install the Loopback Adapter for Windows XP, see [http://support.microsoft.com/kb/839013](http://support.microsoft.com/kb/839013).
E.4 Error Installing RTI DDS Toolkit RT Support

If RTI DDS Toolkit throws errors -8999 or -8998 during installation or uninstallation, the installation/uninstallation has not completed successfully. This can be caused by incorrect permissions. Please rerun either LabVIEW or the VI Package Manager with Administrator privileges.

E.5 Error Using Custom Security Profiles

If you are having problems when trying to create a Reader or Writer while using a Custom Security Profile, make sure that:

- The domainID for the Reader or Writer you want to create is allowed by the permissions file you are loading.
- The topic you want to use is allowed by the permissions file you are loading.
- If you are using an NI Linux RT target, paths to the files to be loaded should point to the NI Linux directory structure and the files must be in the RT target.
E.6 Error Generating ComplexType VIs

If you see error 1154 (shown below) when generating ComplexType VIs, check that the Custom Type Definition (*.ctl) is saved in the same LabVIEW version that the RTI DDS ComplexType Generator is running.

![Error 1154 occurred at invoke Node in Create Simple Reader Custom Type Generator.vi -> RTI DDS ComplexType Generator.vi](image1)

Possible reason(s):
LabVIEW: Cannot replace this object with the type specified.
Method Name: Replace

E.7 RT Device Hangs when Modifying QoS Profiles

If your RT device hangs when modifying the QoS profiles, this may be caused by an incorrect QoS XML file. Some of the possible issues are:

- Malformed XML file
- Incorrect QoS values
- Duplicated QoS profile/library names

![Deployment Progress](image2)