RTI DDS Toolkit

Getting Started Guide

Version 4.0.0

Notices

Deprecations and Removals

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

Deprecated means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported.

Technical Support
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089

Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

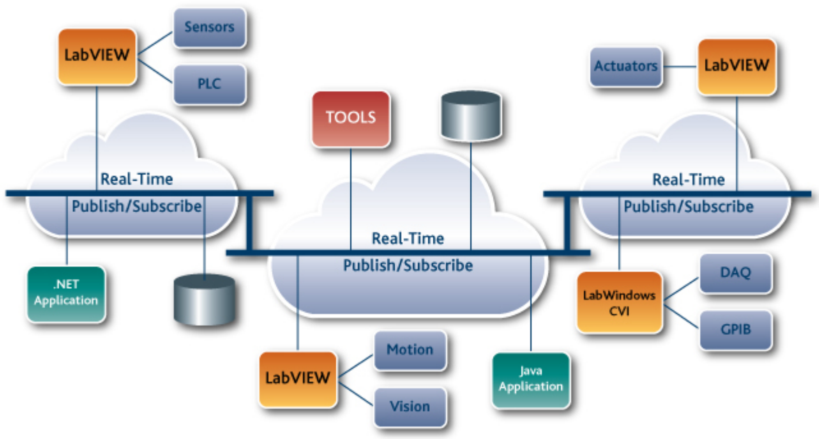
Contents

Chapter 1 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*® and National Instruments® LabVIEW™.

By using LabVIEW and *Connext* 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.

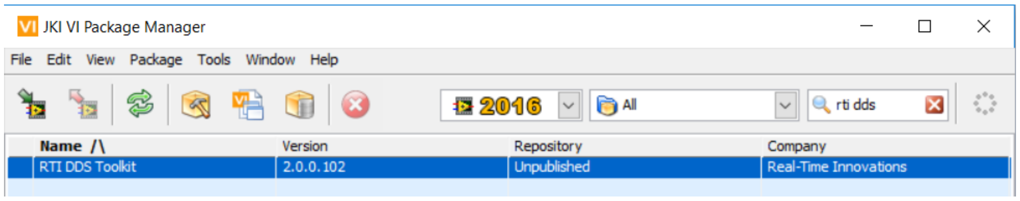
1.2 Installing

Notes:

- If you are upgrading *RTI DDS Toolkit*, skip to 1.4 Upgrading on page 8.
- You need administrator privileges to install the toolkit and to make it available to install in RT targets.

To Install *RTI DDS Toolkit*:

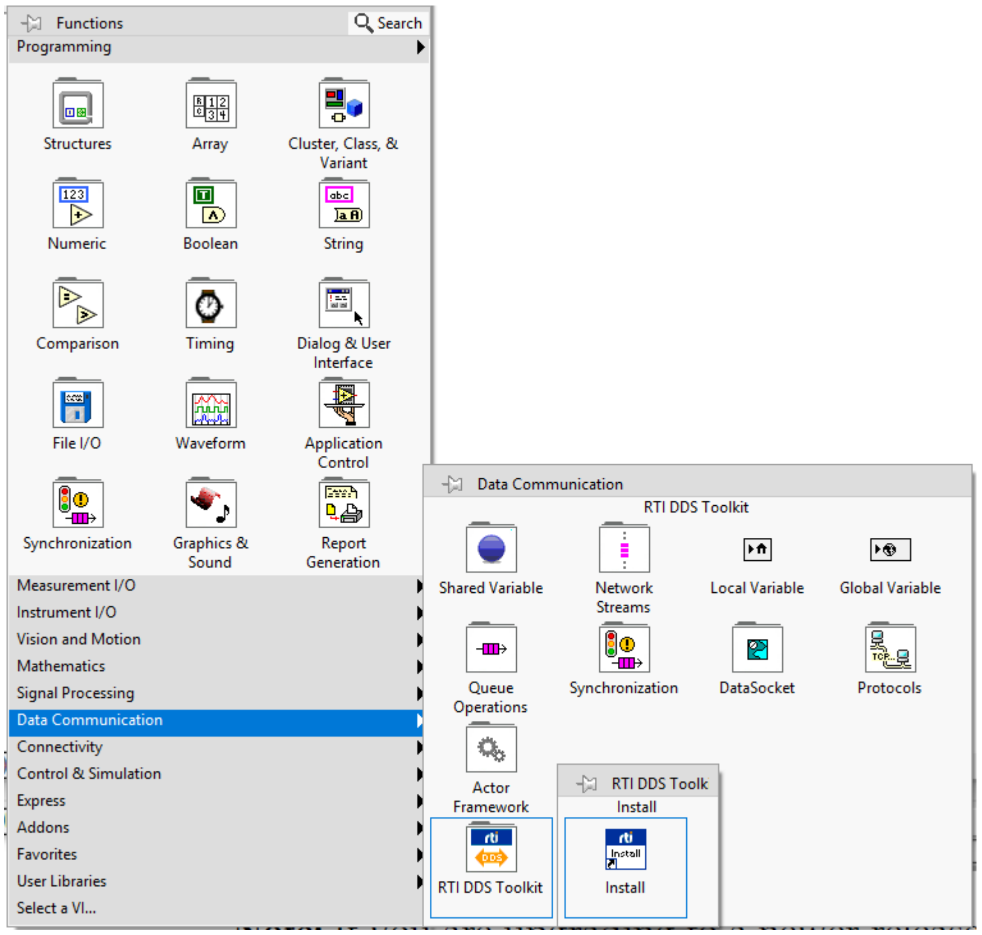
1. Verify you have a supported version of LabVIEW already installed (see the *Release Notes* for supported versions).
2. Login with administrator privileges.
3. Install the JKI LabVIEW VI Package Manager (VIPM) if you have not done so already (available here: http://jki.net/vipm/download). It is typically installed in **C:\Program Files**[1]**\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.**



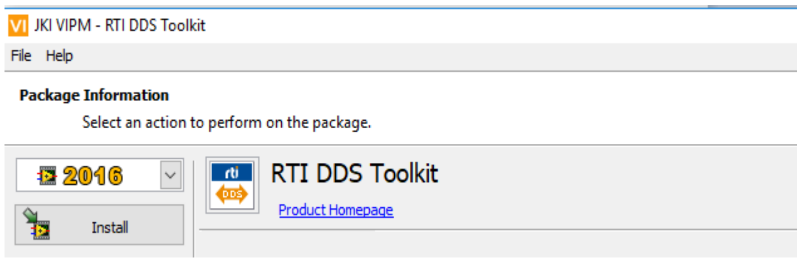
Name /\	Version	Repository	Company
RTI DDS Toolkit	2.0.0.102	Unpublished	Real-Time Innovations

Note: In LabVIEW 2017+, you will find a shortcut from the block diagram: select **Data Communication**, **RTI DDS Toolkit**, **Install**, as seen below:

[1]On 64-bit systems, the folder is "Program Files (x86)"



7. Install *RTI DDS Toolkit*:
    a. Select the LabVIEW version for which you want to install *RTI DDS Toolkit*.



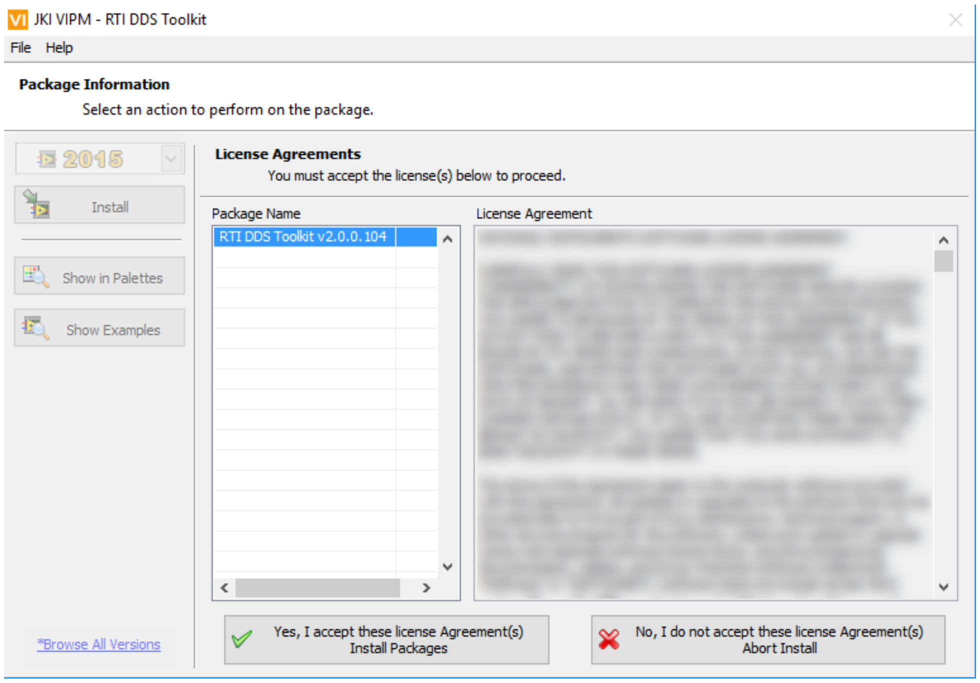
Package Information

Select an action to perform on the package.

    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.


JKI VIPM - RTI DDS Toolkit
File Help
Package Information
Select an action to perform on the package.
2015
Install
Show in Palettes
Show Examples
*Browse All Versions
License Agreements
You must accept the license(s) below to proceed.
Package Name
License Agreement
RTI DDS Toolkit v2.0.0.104
Yes, I accept these license Agreement(s) Install Packages
No, I do not accept these license Agreement(s) Abort Install

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:

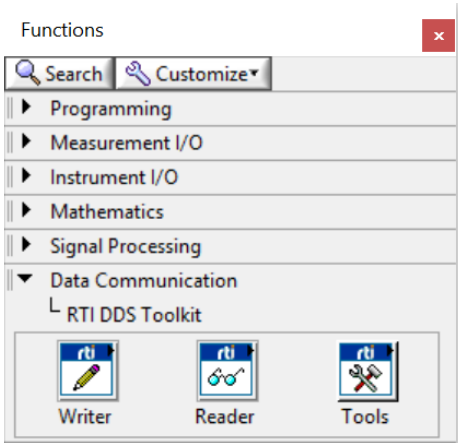
RTI DDS Toolkit support files allow you to deploy VIs using *RTI DDS Toolkit* into your target. The following instructions assume you have JKI VIPM and LabVIEW installed. The following instructions assume you have installed successfully the *RTI DDS Toolkit* which is explained in 1.2 Installing on page 2.

1. Copy **<x86 National Instruments folder>\RT Images\RTI DDS Toolkit\x.x.x\/rti-dds-labviewtoolkit_x.x.x_x64.ipk** to the **/home/admin** folder on the RT system using FTP or SCP.
2. SSH into the target, then run **opkg install rti-dds-toolkit-x.x.x.x_x64.ipk**.
3. Reboot the RT 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**.

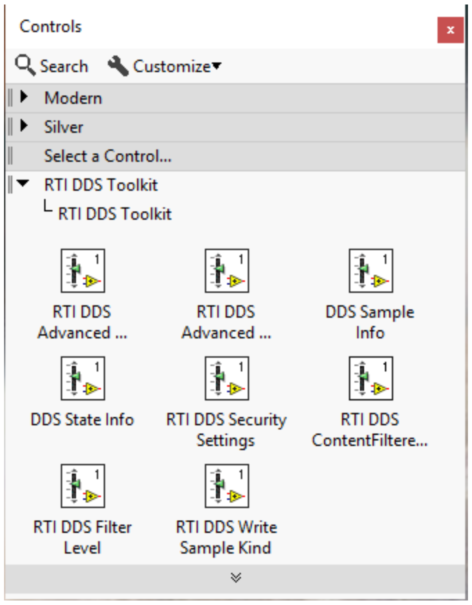
   For details, see 1.3.1 LabVIEW Functions Palette on the next page.



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 1.3.2 LabVIEW Controls Palette on page 8.

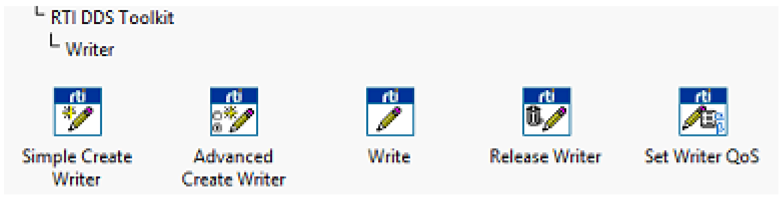
   See also: Appendix D File Folders Installed within LabVIEW on page 153.



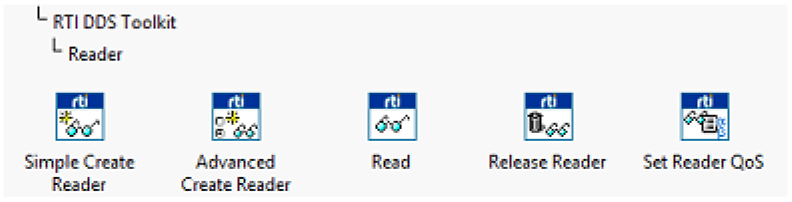
1.3.1 LabVIEW Functions Palette

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

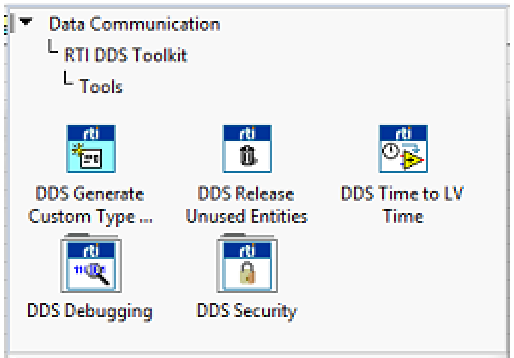
- **RTI DDS Toolkit**
  - **Writer**
    - Simple Create Writer
    - Advanced Create Writer
    - Write
    - Release Writer
    - Set Writer QoS

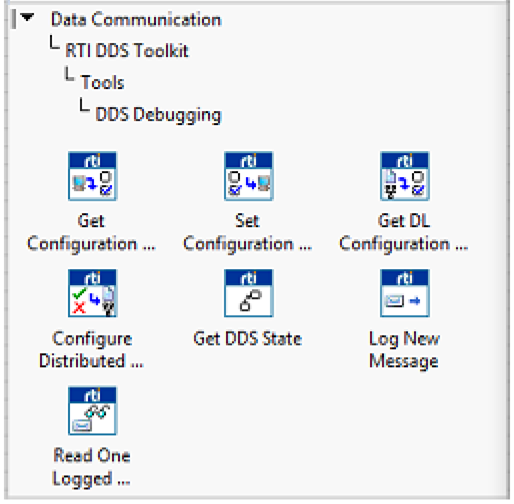


  - **Reader**
    - Simple Create Reader
    - Advanced Create Reader
    - Read
    - Release Reader
    - Set Reader QoS

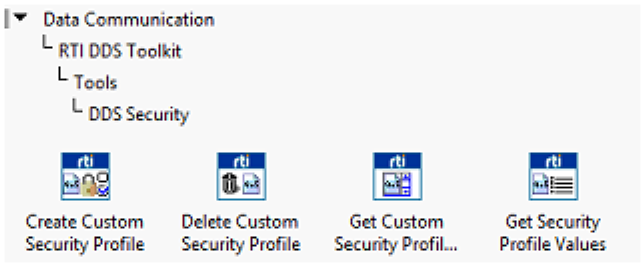


- **Tools**
  - DDS Generate Custom Type VIs
  - DDS Release Unused Entities
  - DDS Time to LV Time
  - DDS Debugging
    - Get Configuration Parameters
    - Set Configuration Parameters
    - Get DL Configuration Parameters
    - Configure Distributed Logger
    - Get DDS State
    - Log New Message
    - Read One Logged Message





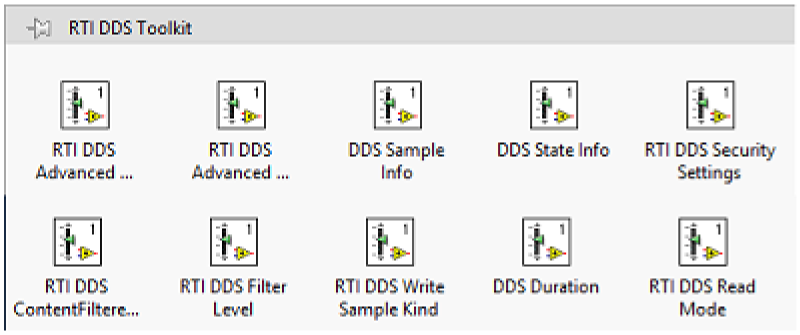
  - DDS Security
    - Create Custom Security Profile
    - Delete Custom Security Profile
    - Get Custom Security Profile
    - Get Security Profile Values


Data Communication
RTI DDS Toolkit
Tools
DDS Security
Create Custom Security Profile
Delete Custom Security Profile
Get Custom Security Profil...
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
  - DDS Duration
  - RTI DDS Read Mode



1.4 Upgrading

If you have already installed *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 *DDS Toolkit* **.vip** file.
4. Upgrade *DDS Toolkit*:
   a. Select the LabVIEW version for which you want to upgrade *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 *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 Version before 3.1.2

DDS Toolkit 3.2.0 is a single monolithic library called **rtilvdds.dll**. It contains a standard OpenSSL distribution and all the *RTI Connext DDS* dependencies (nddscore, nddsc, rtimonitoring, nddssecurity, and rtidlc libraries). In previous versions, RTI dependencies were shipped as separate libraries (nddsc.dll, nddscore.dll, nddssecurity.dll, rtidlc.dll, and rtimonitoring.dll). Because of this change:

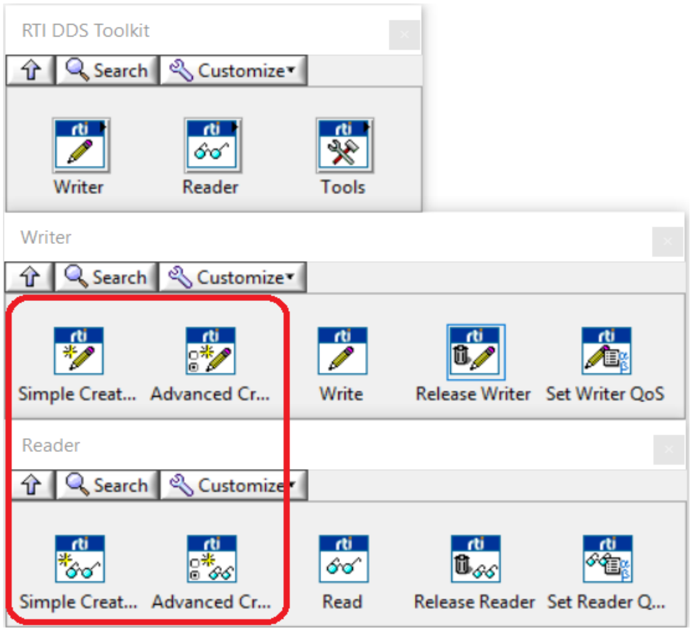
- Nddssecurity now uses standard OpenSSL 1.1.1t instead of using the NI SSL included in LabVIEW.
- Rtimonitoring and nddssecurity are now linked statically, so the way they are enabled has changed. However, this change only affects QoS files; no need to change the code.
- When shipping standalone applications, there is no need to add dependencies. Since the rtilvdds library is a first level dependency, LabVIEW should pick it automatically.
- Security is not supported in cRIO targets. Before updating to 3.1.2, disable security in the cRIO.
- ARM cRIOs are not supported.

Refer to 4.9.2 Adapting a VI to Use RTI Monitoring Library and 6.8 Enabling Security (Windows only) in this document for more information about the new usage.

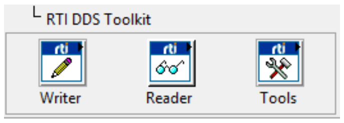
1.4.2 Additional Steps when Upgrading from a Version Before 2.0.0.104

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.

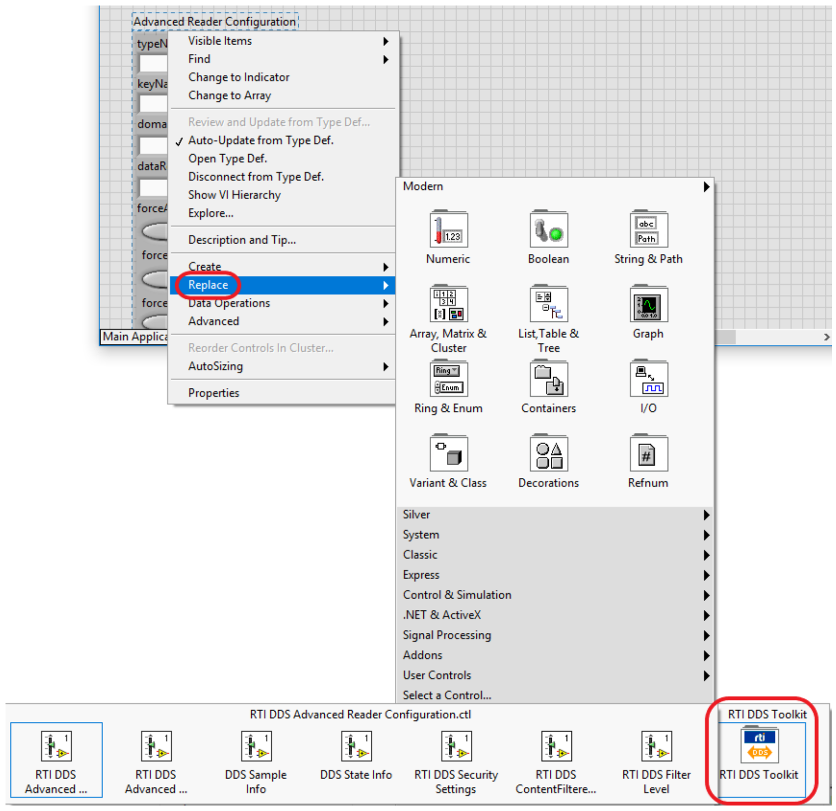
- The create Reader/Writer subVIs have been removed. Use the create simple/advanced Reader/Writer subVIs instead. These VIs are included in the RTI DDS Toolkit/Reader and RTI DDS Toolkit/Writer subpalettes.



- The Complex-Type Templates are no longer supported. Therefore, that subpalette has been removed. Instead. Direct CLF calls are no longer supported; use the VIs generated using the *RTI DDS ComplexType Generator* instead. See 6.3.1 Using the RTI DDS ComplexType Generator on page 99.



- 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.



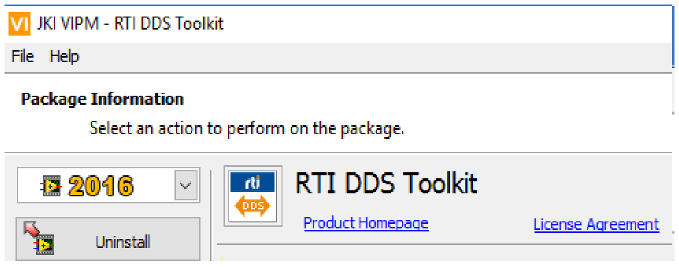
.

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.



Package Information

Select an action to perform on the package.

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

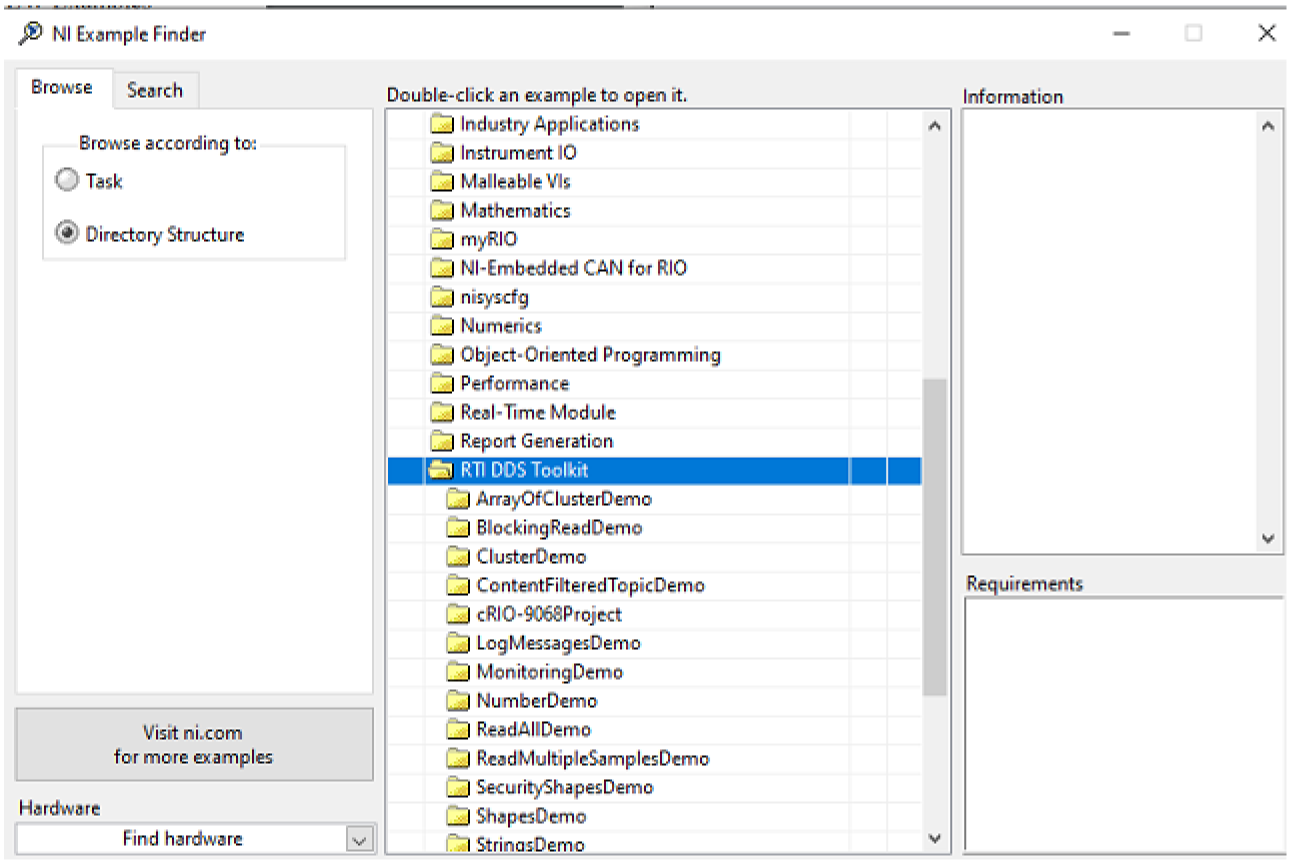
The toolkit should be uninstalled using the opkg tool on the target:

1. SSH into the RT target as admin (or another user with administrator privileges).
   a. Enter **ssh <target ip>@admin**.
2. Uninstall the *DDS Toolkit* support files using opkg.
   a. Enter **opkg remove rti-dds-labview-toolkit**.

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:

- **ArrayOfClusterDemo**: Shows how to read and write a type with arrays of clusters.
- **BlockingReadDemo**: Shows how to perform a read operation in blocking read mode. See 4.3 Lesson 3—Blocking Reads on page 45.
- **ClusterDemo**: Shows how to handle complex types (such as clusters). It was created by following the lessons in Chapter 4 Tutorial on page 29:
  - 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38
  - 4.4 Lesson 4—Filtering Data on page 47
  - 4.5 Lesson 5—Reading Only New Samples on page 52
- **ContentFilteredTopicDemo:** shows how to filter data using a ContentFilteredTopic. It has been created by following 4.4.2 Filtering Data Using ContentFilteredTopics on page 49.
- **cRIOProject**: Shows how to use *RTI DDS Toolkit* on a cRIO 9068. See 4.10 Lesson 10—Using RTI DDS Toolkit on NI Targets (cRIO-9068 Example) on page 79.
- **LogMessagesDemo**: Shows how to log debugging messages into the internal queue. It was created by following 4.9.1.1 Logging Messages Manually on page 74.

- **MonitoringDemo**: Uses a QoS profile that enables *RTI Monitoring Library*. It was created by following 4.9.2 Adapting a VI to Use RTI Monitoring Library on page 76.
- **NumberDemo**: Shows how to read and write a simple type (such as a numeric one). It was created by following 4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) on page 30.
- **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. See 4.8 Lesson 8—Reading All Samples (Reliable Communication) on page 64.
- **ReadMultipleSamplesDemo**: Shows how to read multiple samples in a single call. See 4.12 Lesson 12—Reading Multiple Samples at a Time on page 88.
- **SecurityShapesDemo**: This example will show how to use different security profiles and how they behave depending on the permissions they have. See 4.11 Lesson 11—Using Security with RTI DDS Toolkit (Windows only) on page 83.
- **ShapesDemo**: Shows how to publish and subscribe to an already existing DDS application: *RTI Shapes Demo*. See 4.6 Lesson 6—Using Keyed Types (RTI Shapes Demo) on page 55.
- **StringsDemo**: Shows how to write a string. See Chapter 3 A Simple Read/Write Example on page 21.

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 3.4.2 Preventing 'Type Code Incorrect' Error when Working with Arrays on page 27.

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 patter 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.

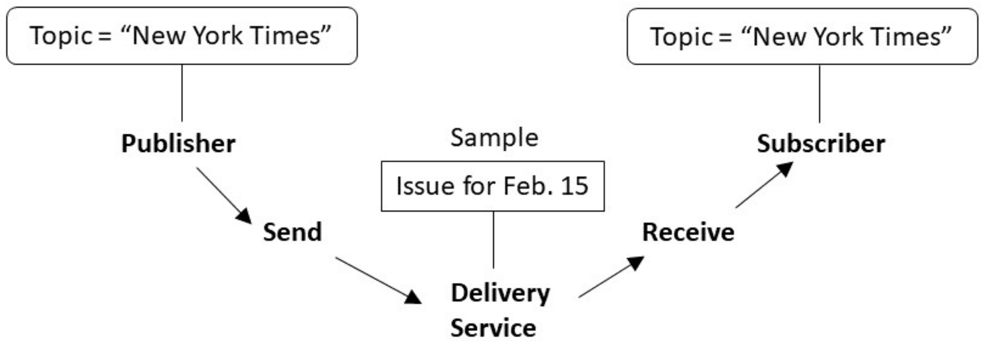
- Publish/subscribe, by providing both discovery and messaging, decouples the producers and consumers effectively. DCPS middleware provides publish/subscribe services to an application-defined data-model, allowing fine-grained control of QoS, enabling the infrastructure to do smart-caching of the information 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 delivers 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 An Example of Publish-Subscribe below.

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



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 inherent from them. For your convenience, you can find an XML file that shows you these profiles in **C:/Program Files[1]/National Instruments/LabVIEW 20xx/vi.lib/_RTI DDS Toolkit_internal_deps/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.

On RTI's Community Forum (http://community.rti.com), you can find more information about QoS properties and XML configuration, as well as the XSD schema.

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.

[1]On 64-bit systems, the folder is "Program Files (x86)"

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 provide 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 air planes, 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 on page 29 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.

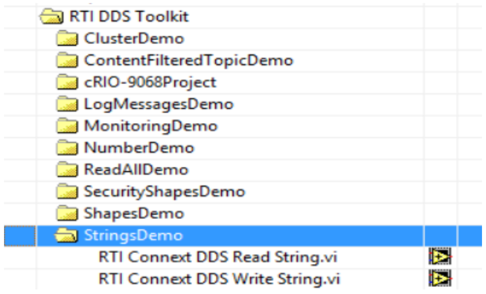
Before continuing, please make sure you have the following software installed:

- 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 E.3 Running without an Active Network Interface on page 165.

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.

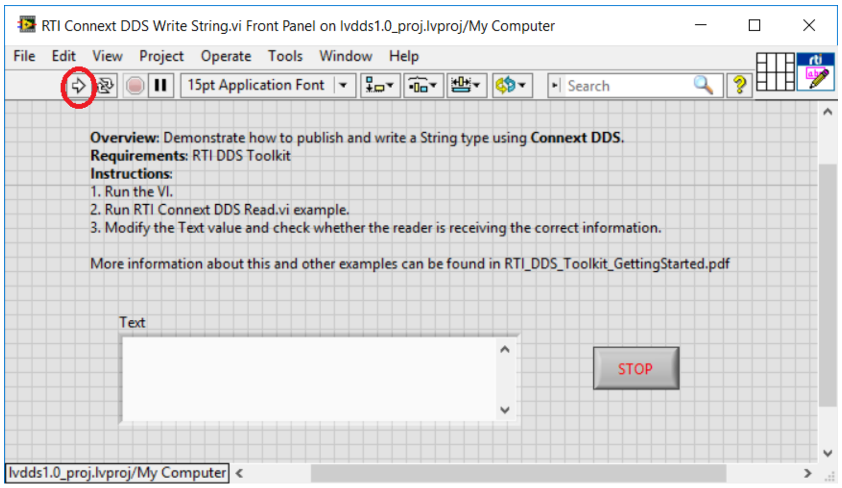
5. Open the **StringsDemo** folder.

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 3.4.3 Troubleshooting with Ping and Spy on page 27.
- 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. 3.4 Usage Notes on page 26 explains how to resolve this.

3.1 Publishing a String in DDS

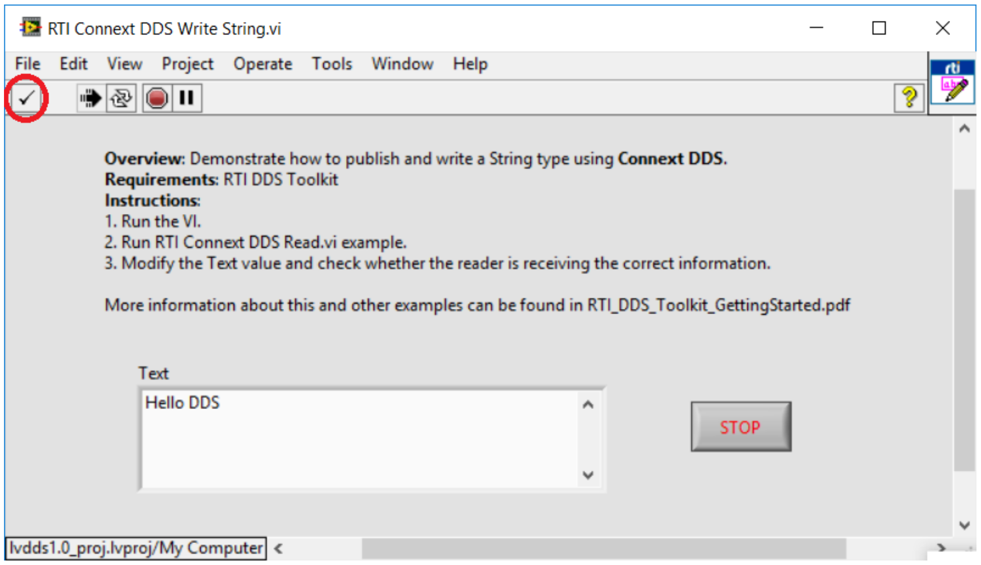
1. Open the **RTI Connext DDS Write String.vi** by double-clicking on it in the NI Example Finder (select **Help, Find Examples...**).
2. Click the **Run** button in the LabVIEW toolbar.


RTI Connext DDS Write String.vi Front Panel on lvdds1.0_proj.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Text
STOP
lvdds1.0_proj.lvproj/My Computer

Overview: Demonstrate how to publish and write a String type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Read.vi example.
3. Modify the Text value and check whether the reader is receiving the correct information.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

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.


RTI Connext DDS Write String.vi
File Edit View Project Operate Tools Window Help
STOP
lvdds1.0_proj.lvproj/My Computer

Overview: Demonstrate how to publish and write a String type using **Connext DDS**.
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Read.vi example.
3. Modify the Text value and check whether the reader is receiving the correct information.

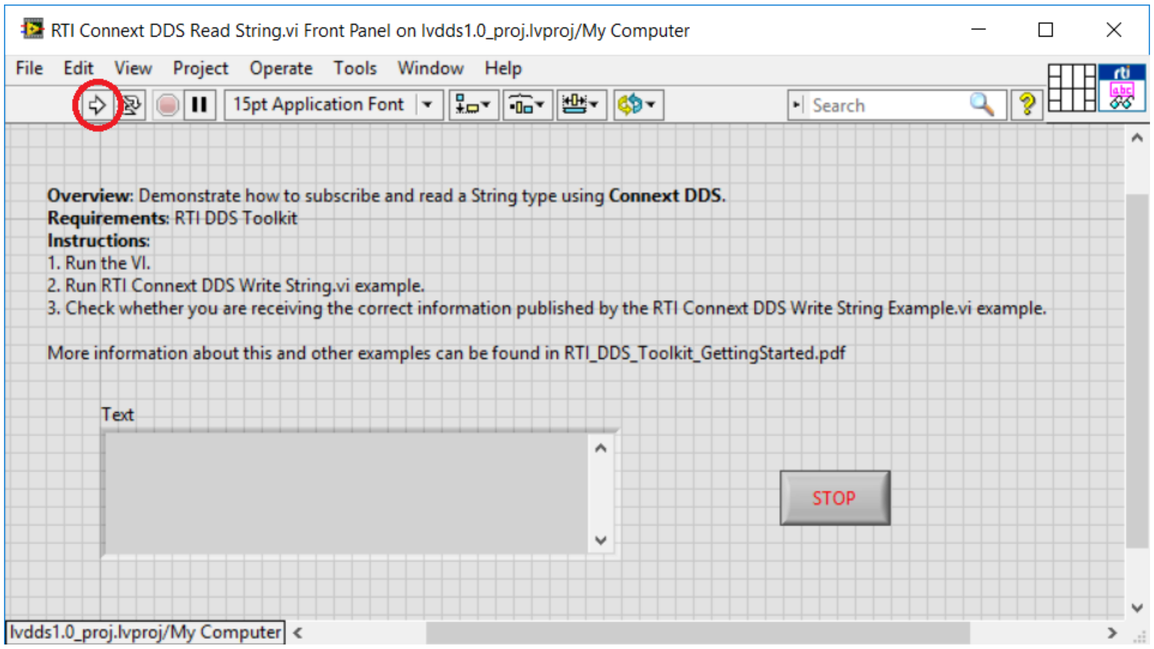
More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Text
Hello DDS

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

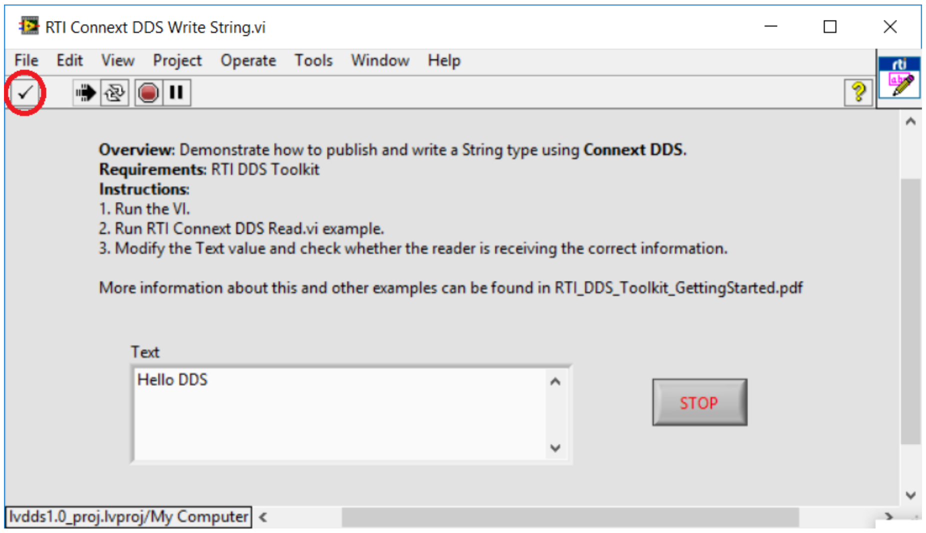
1. Open the **RTI Connext DDS Read String.vi** by double-clicking on it in the NI Example Finder (select **Help, Find Examples...**).
2. Click on the **Run** ⇨ button in the LabVIEW toolbar.


RTI Connext DDS Read String.vi Front Panel on lvdds1.0_proj.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Text
STOP
lvdds1.0_proj.lvproj/My Computer

Overview: Demonstrate how to subscribe and read a String type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Write String.vi example.
3. Check whether you are receiving the correct information published by the RTI Connext DDS Write String Example.vi example.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

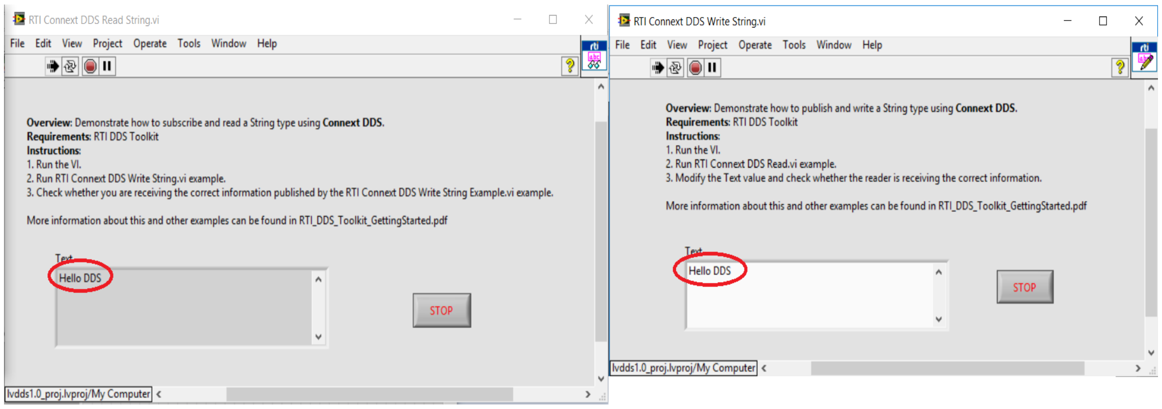
3. Verify that it is reading the same string that is being published from the **RTI Connext DDS Write String.vi**.


RTI Connext DDS Write String.vi
File Edit View Project Operate Tools Window Help
Text
STOP
lvdds1.0_proj.lvproj/My Computer

Overview: Demonstrate how to publish and write a String type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Read.vi example.
3. Modify the Text value and check whether the reader is receiving the correct information.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Hello DDS


RTI Connext DDS Read String.vi
File Edit View Project Operate Tools Window Help
Text
Hello DDS
STOP
lvdds1.0_proj.lvproj/My Computer
RTI Connext DDS Write String.vi
File Edit View Project Operate Tools Window Help
Text
Hello DDS
STOP
lvdds1.0_proj.lvproj/My Computer

Overview: Demonstrate how to subscribe and read a String type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Write String.vi example.
3. Check whether you are receiving the correct information published by the RTI Connext DDS Write String Example.vi example.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Overview: Demonstrate how to publish and write a String type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Read.vi example.
3. Modify the Text value and check whether the reader is receiving the correct information.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

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?

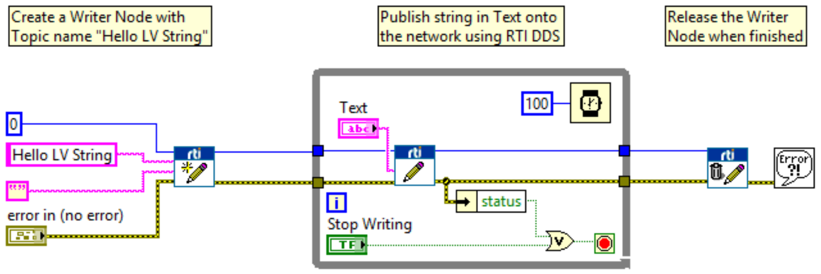
To better understand how this demonstration is implemented, let's review the code for these two VIs:

- 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 A.2.1 Writer on page 135.

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:

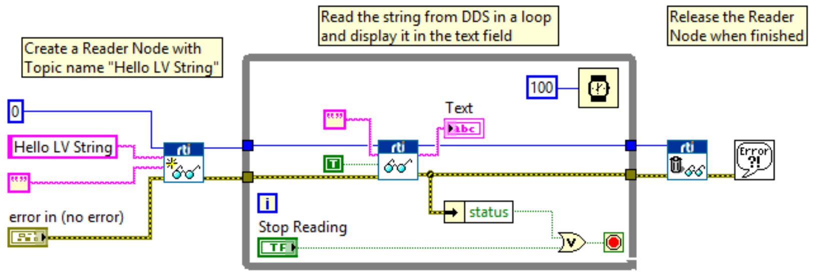

Create a Writer Node with Topic name "Hello LV String"
Publish string in Text onto the network using RTI DDS
Release the Writer Node when finished
0
Hello LV String
error in (no error)
Text
100
status
Stop Writing

- 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 A.2.2 Reader on page 137.

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:


Read the string from DDS in a loop and display it in the text field
Release the Reader Node when finished
Create a Reader Node with Topic name "Hello LV String"
0
Hello LV String
error in (no error)
100
Text
status
Stop Reading

3.4 Usage Notes

3.4.1 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 Configuratio*n* flag **forceUnboundedString** to **true**, they are created with a length equivalent to the maximum integer (2,147,483,647) (see 4.7 Lesson 7—Used Nested and Multiple Keys on page 61). 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:

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

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.

3.4.2 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.

3.4.3 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[1].

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*[2] (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.

RTI Monitor is included in *RTI Connext DDS Professional*. You can download a free trial from http://www.rti.com/downloads/index.html. For information about *RTI Monitor*, see https://www.rti.-com/products/dds/tools#MONITOR.

[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 2020 (32-bit) or later for Windows systems
- *RTI DDS Toolkit* for National Instruments LabVIEW 2020 (32-bit) or higher for Windows systems

The tutorial includes these lessons:

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 4.13 Reviewing Completed Solutions on page 92.

Notes:

- These lessons assume you are familiar with LabVIEW.
- For debugging information, see E.1 Enabling Debugging Mode on page 155

4.1 Lesson 1–Using DDS to Publish and Subscribe to Simple Data (Numeric)

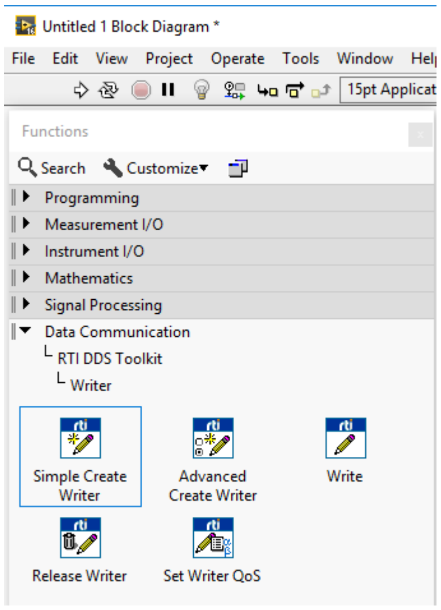
In this first lesson, you will become familiar with the *RTI DDS Toolkit* functions and capabilities by creating two LabVIEW VIs that can publish and subscribe to data. You can run these VIs on the same computer or separate computers connected to the same local area network. *RTI DDS Toolkit* will automatically 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 A.2.1 Writer on page 135.

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:

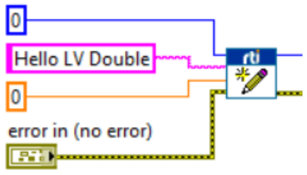
a. Right-click on the *Create Writer* subVI and select **Select Type, Numeric (DBL)**

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 = Hello LV 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.

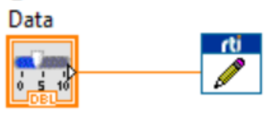
1. Open the Functions Palette and select **Data Communication, RTI DDS Toolkit, Writer, Write**; drag and drop the *Write* subVI into the Block Diagram.

   The *Write* subVI has the following input parameters:

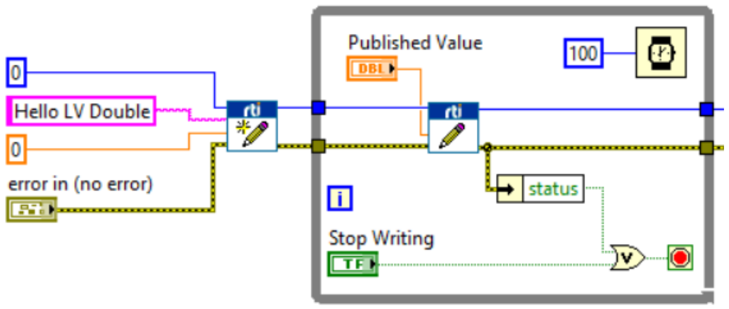
   - DDS Object Ref in
   - Data
   - error in
     For details on these parameters, see A.2.1 Writer on page 135.

2. Wire the **DDS Object Ref** output of the *Create Writer* subVI (from 4.1.1.1 Create a Writer Object to Publish a Numeric (DBL) on the previous page) 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:



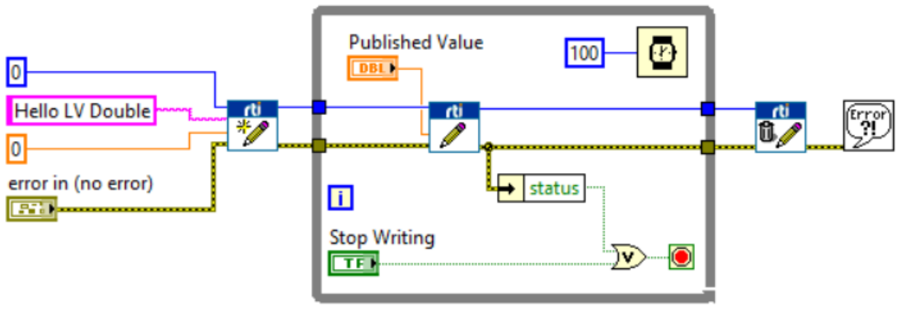
4.1.1.3 Release the Writer Object

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 A.2.1 Writer on page 135.

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



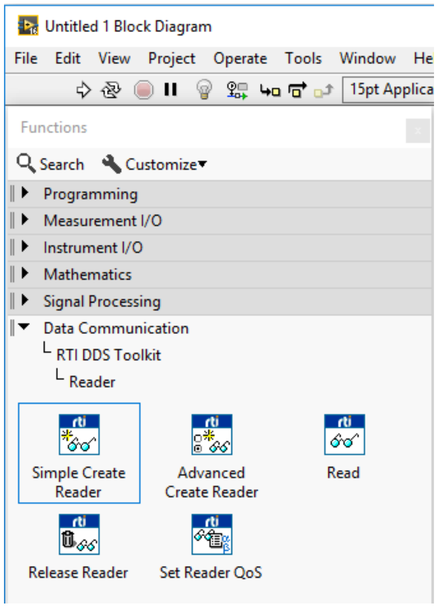
3. Save the file **Tutorial_Write_Double.vi**.
4. We recommend including error handling in your VIs. Take the above figure as an example: we use error handler's *status* to control the loop exit condition.

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

In 4.1.1 Developing a VI to Publish Simple Data (Numeric) on page 30, 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 2016, 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 A.2.2 Reader on page 137.

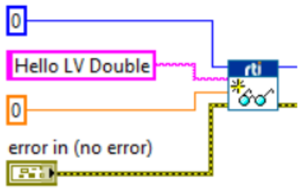
   We will use this subVI to create a *Reader* object that can subscribe to a data type of Numeric (DBL). We will use domain ID 0 and our Topic Name will be **Hello LV Double**. To begin:

   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:



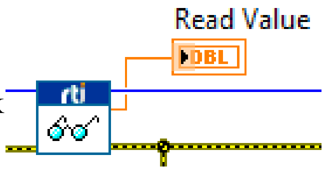
4.1.2.2 Subscribe to a Numeric (DBL)

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)
    - Read Mode (Optional. Default is Polling Mode.)
    - Blocking Timeout (Optional. Default is 0 seconds and 0 nanoseconds.)

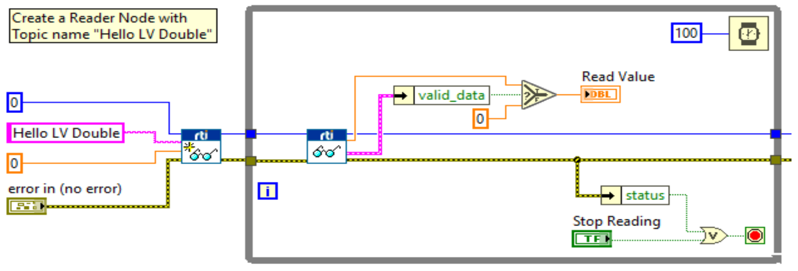
    For details on these parameters, see A.2.2 Reader on page 137.
2. Wire the *Create Reader* subVI's **DDS Object Ref** output node to the *Read* subVI's **DDS Object Ref in** input node.
3. In this example, we will subscribe to the Numeric (DBL) published by the **Tutorial_Write_Double.vi**. To display 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. (Optional) Select a read operation mode and blocking timeout. For details on these parameters, see A.2.2 Reader on page 137.
6. 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

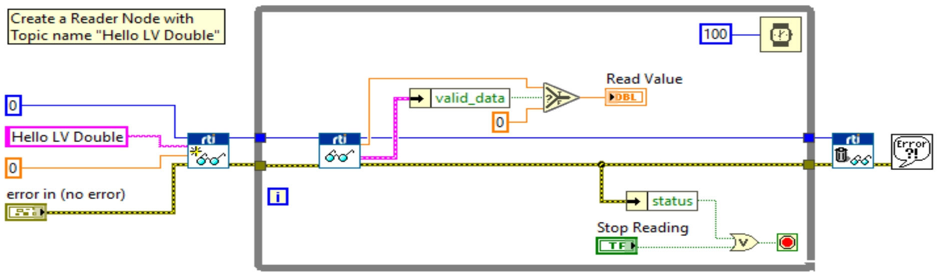
The final step in our **Tutorial_Read_Double.vi** is to release the DDS entities and reclaim the system resources when the *While Loop* execution is terminated. To do this, we use the *Release Reader* subVI in the Block Diagram.

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 A.2.2 Reader on page 137.

   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:

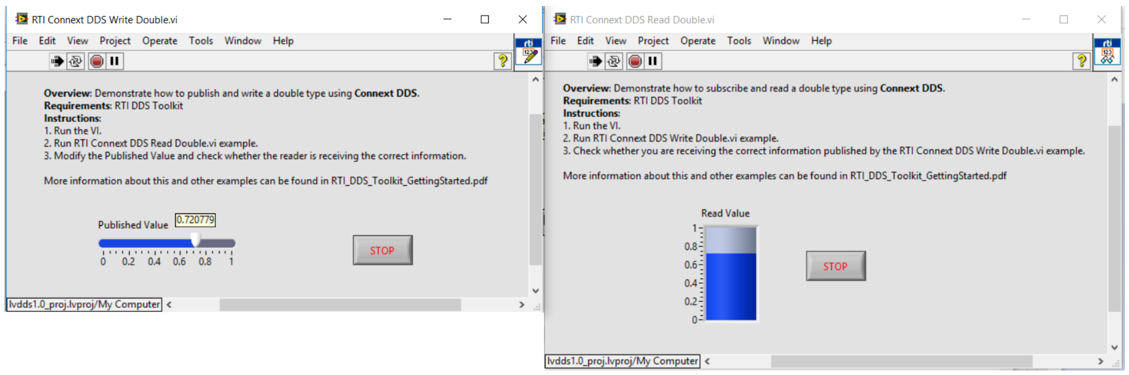


3. Save the file **Tutorial_Read_Double.vi**.
4. We recommend including error handling to your VIs. Please see 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on the next page and 4.4 Lesson 4—Filtering Data on page 47 for further details.

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**.


RTI Connext DDS Write Double.vi
File Edit View Project Operate Tools Window Help
Published Value
0.720779
0 0.2 0.4 0.6 0.8 1
STOP
lvdds1.0_proj.lvproj/My Computer
RTI Connext DDS Read Double.vi
File Edit View Project Operate Tools Window Help
Read Value
1 0.8 0.6 0.4 0.2 0
STOP
lvdds1.0_proj.lvproj/My Computer

Overview: Demonstrate how to publish and write a double type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Read Double.vi example.
3. Modify the Published Value and check whether the reader is receiving the correct information.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Overview: Demonstrate how to subscribe and read a double type using **Connext DDS**.
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Write Double.vi example.
3. Check whether you are receiving the correct information published by the RTI Connext DDS Write Double.vi example.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

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.’

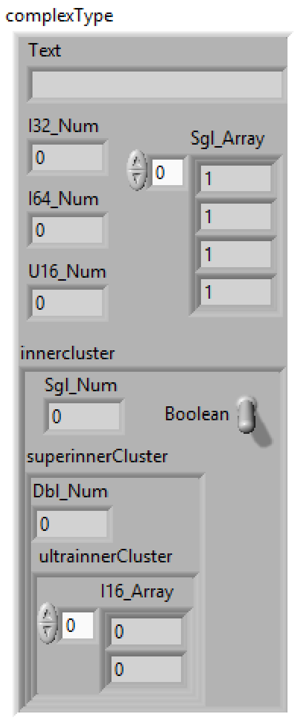
4.2 Lesson 2–Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters)

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 on the enum and select Representation—>32.

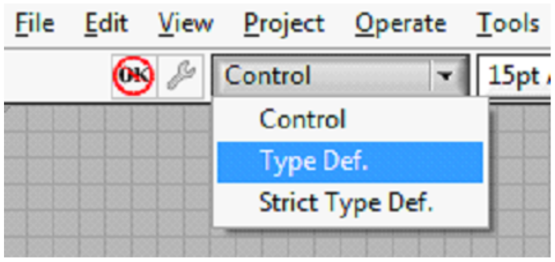
We are going to focus on the cluster use-case. Let's begin by developing a VI that can publish the cluster defined in the following figure:

Figure 4.1 Complex Type

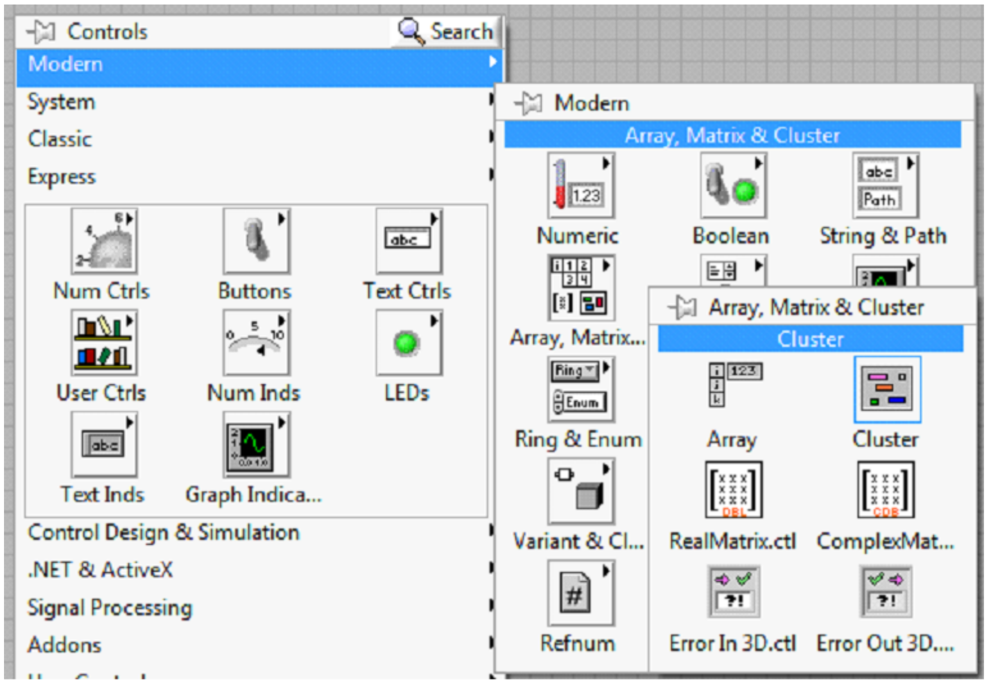


First, we will define a new type (a LabVIEW Type-Def) for this cluster:

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 3.4.2 Preventing 'Type Code Incorrect' Error when Working with Arrays on page 27.

4. Fill the **complexType** cluster as shown in Figure 4.1 Complex Type on page 38. This process is simple: drag the following controls from the Palette:

    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. See 6.10 Setting Up Arrays on page 129.

    f. Cluster inside the first one labeled as **innercluster.** Fill **innercluster** as shown in Figure 4.1 Complex Type on page 38.

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 6.3.1 Using the RTI DDS ComplexType Generator on page 99 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: **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.

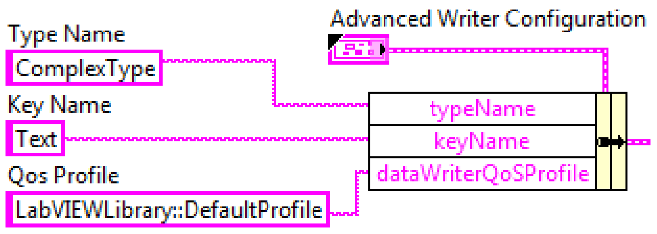
Several new VIs will be created in the output directory:

- **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**
- **Tutorial_Cluster Multiple_Samples.vi**

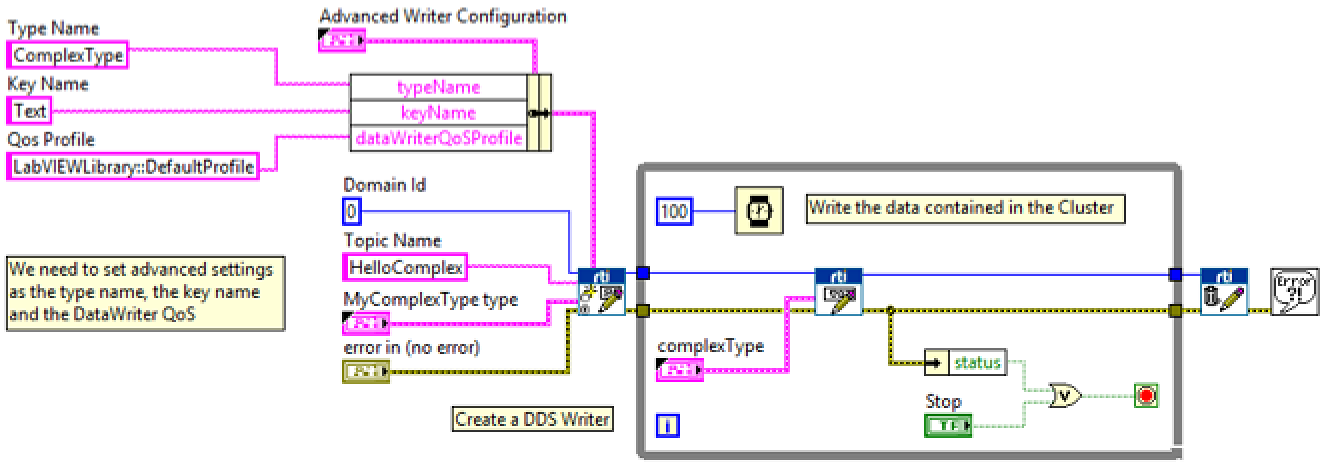
4.2.1.1 Modify the Writer Example VI

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**.



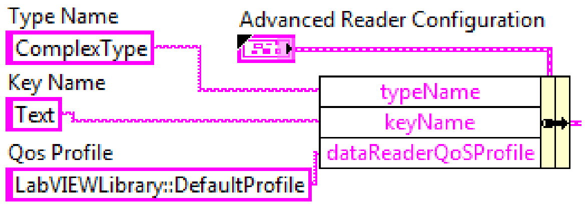
Note: For details on Advanced Settings, see Chapter 6 Advanced Concepts and Settings on page 96.The resulting Block Diagram should look similar to this:



5. Save the file **Tutorial_Cluster Writer Example.vi**.

4.2.1.2 Modify the Reader Example VI

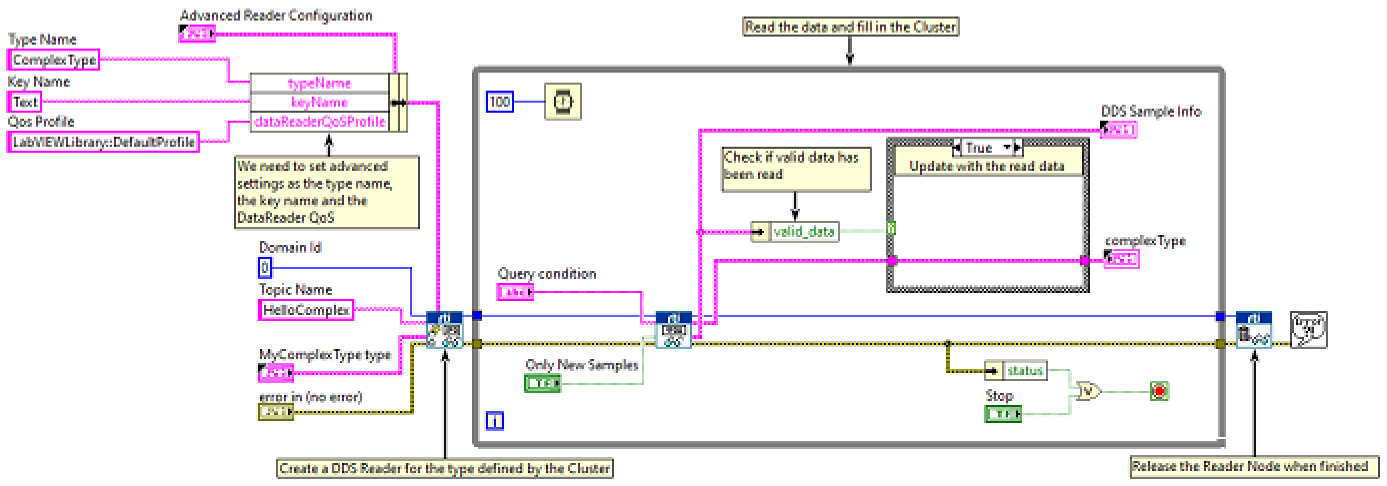
Open the **Tutorial_Cluster Reader Example.vi** and change the Advanced Reader Configuration control the same way explained in 4.2.1.1 Modify the Writer Example VI on the previous page.



Note: For details on Advanced Settings, see Chapter 6 Advanced Concepts and Settings on page 96.

1. Delete the ContentFilteredTopic Info control, since we are not going to use it in this lesson (See 4.4.2 Filtering Data Using ContentFilteredTopics on page 49).
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:


Advanced Reader Configuration
Type Name
ComplexType
Key Name
Text
Qos Profile
LabVIEWLibrary::DefaultProfile
typeName
keyName
dataReaderQoSProfile
We need to set advanced settings as the type name, the key name and the DataReader QoS
Domain Id
0
Topic Name
HelloComplex
MyComplexType type
error in (no error)
Create a DDS Reader for the type defined by the Cluster
Read the data and fill in the Cluster
100
Check if valid data has been read
valid_data
True
Update with the read data
DDS Sample Info
complexType
Query condition
Only New Samples
status
Stop
Release the Reader Node when finished

3. Save the file **Tutorial_Cluster Reader Example.vi**.

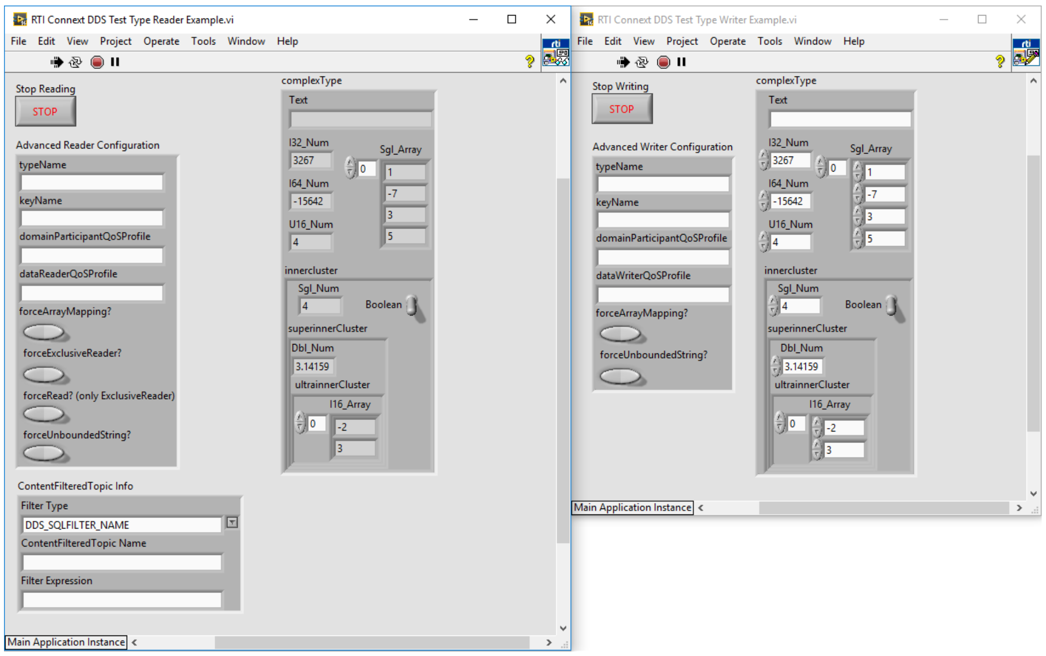
4.2.1.3 Creating VIs Programmatically

Custom ComplexType VIs can also be created programmatically using the VI named *DDS Generate Custom Type VIs* under the Tools palette. The input parameters are the same ones used in ComplexType Generator (see 6.3.1 Using the RTI DDS ComplexType Generator on page 99). The output parameter is the error cluster and a path array with paths to the generated files.

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**.


RTI Connext DDS Test Type Reader Example.vi
File Edit View Project Operate Tools Window Help
Stop Reading
STOP
RTI Connext DDS Test Type Writer Example.vi
File Edit View Project Operate Tools Window Help
Stop Writing
STOP
Main Application Instance

Advanced Reader Configuration
typeName
keyName
domainParticipantQoSProfile
dataReaderQoSProfile
forceArrayMapping?
forceExclusiveReader?
forceRead? (only ExclusiveReader)
forceUnboundedString?

ContentFilteredTopic Info
Filter Type: DDS_SQLFILTER_NAME
ContentFilteredTopic Name
Filter Expression

complexType
Text
I32_Num: 3267
I64_Num: -15642
U16_Num: 4
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 4
Boolean
superinnerCluster
Dbl_Num: 3.14159
ultrainnerCluster
I16_Array: 0 | -2, 3

Advanced Writer Configuration
typeName
keyName
domainParticipantQoSProfile
dataWriterQoSProfile
forceArrayMapping?
forceUnboundedString?

complexType
Text
I32_Num: 3267
I64_Num: -15642
U16_Num: 4
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 4
Boolean
superinnerCluster
Dbl_Num: 3.14159
ultrainnerCluster
I16_Array: 0 | -2, 3

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*.

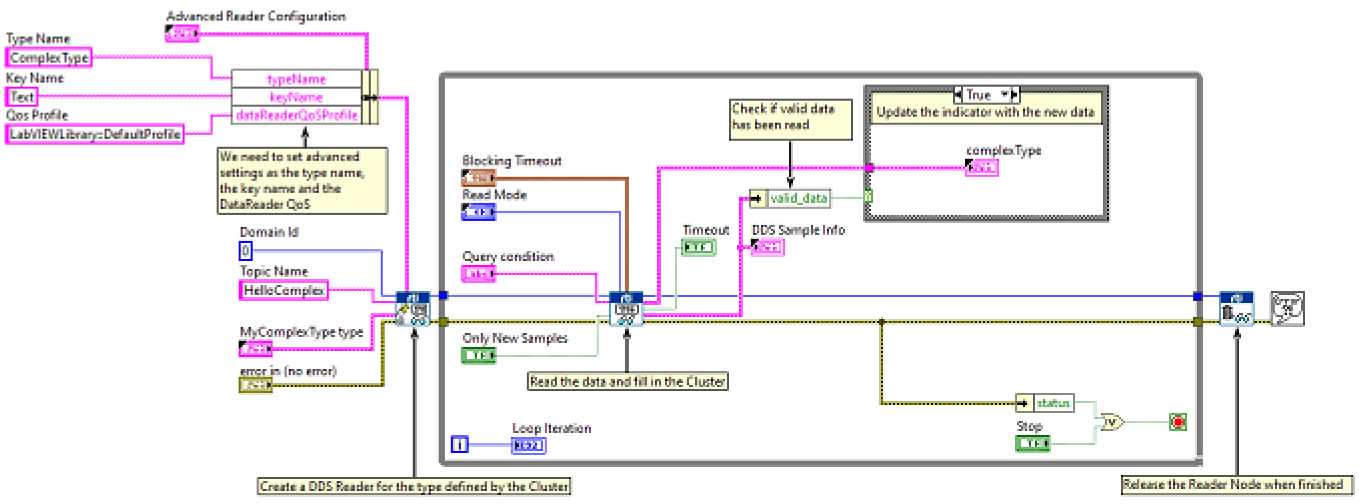
   You will see it flicker on the *Reader* side between the previous and current values. This is the expected behavior because **Text** is our cluster's key. This means that a new instance is created for each **Text** value provided. Even after reading the sample, the received instance is still alive, so it can be reached from the *Reader*. See 4.5 Lesson 5—Reading Only New Samples on page 52 to learn more about this.

4.3 Lesson 3–Blocking Reads

In this lesson you will learn how to use the *Read* VI in *blocking* read operation mode (instead of *polling*). When performing a blocking read, the process waits an amount of time (passed as a parameter) until a sample can be read. This saves CPU cycles because this way it is not necessary to continuously poll for samples.

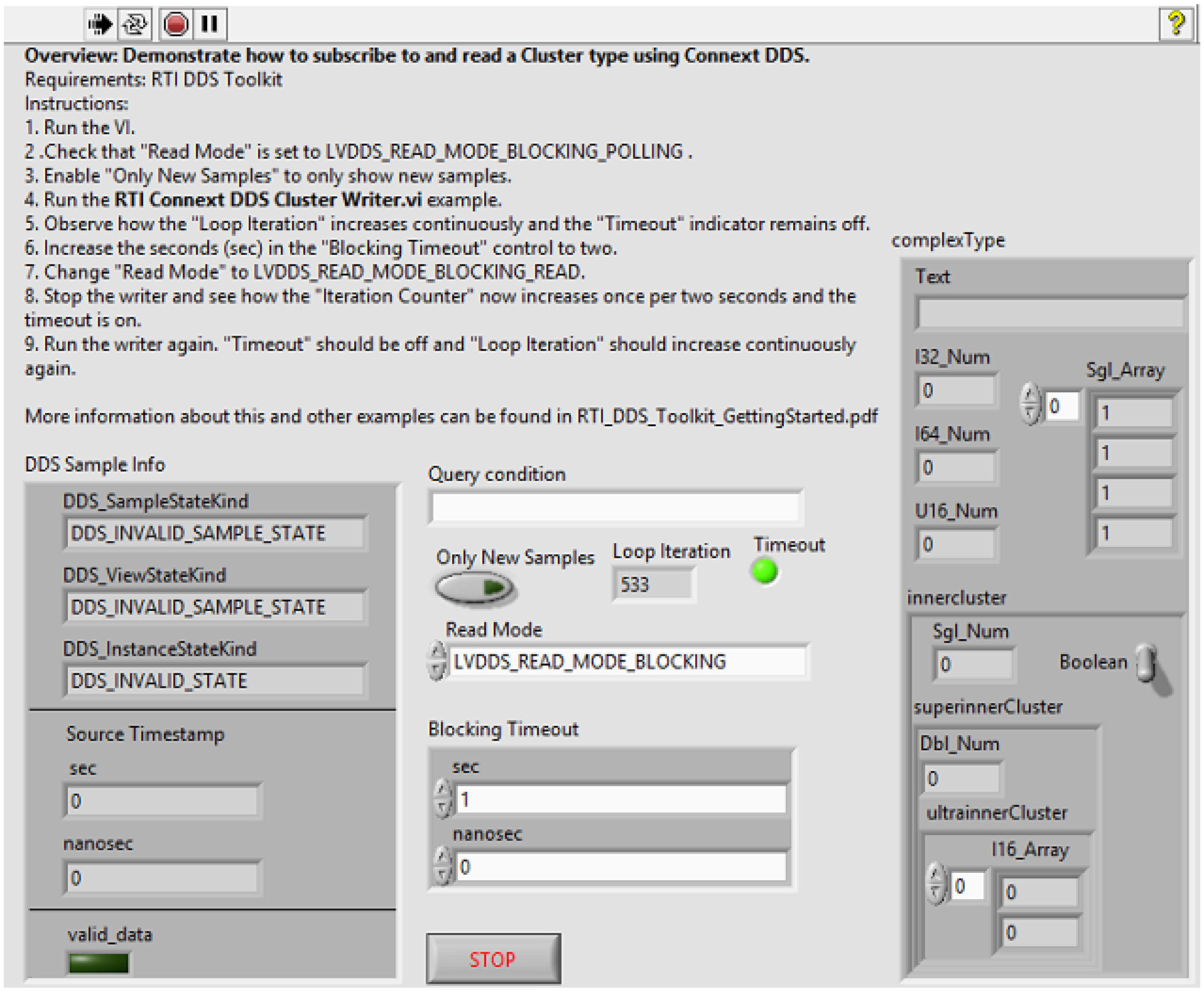
This lesson assumes you have completed 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38.

1. Open the **Tutorial_Cluster Reader Example** from 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38 and save it as a new VI named **Tutorial_BlockingRead_Cluster.vi**.
2. Delete the *Wait* VI inside the loop. We'll use the *Read* VI to wait.
3. Create an integer indicator and wire it to the loop iteration counter. This will help us see how fast loop iterations are made.
4. Set the input "Read Mode" to LVDDS_READ_MODE_BLOCKING.
5. Set "Blocking Timeout" to how long you want to remain blocked until a sample is read (such as 1 second and 0 nanoseconds).


Advanced Reader Configuration
Type Name
ComplexType
Key Name
Text
Qos Profile
LabVIEWLibrary::DefaultProfile
typeName
keyName
dataReaderQoSProfile
We need to set advanced settings as the type name, the key name and the DataReader QoS
Domain Id
0
Topic Name
HelloComplex
MyComplexType type
error in (no error)
Blocking Timeout
Read Mode
Query condition
Only New Samples
Read the data and fill in the Cluster
Loop Iteration
Timeout
DDS Sample Info
Check if valid data has been read
valid_data
True
Update the indicator with the new data
complexType
status
Stop
Create a DDS Reader for the type defined by the Cluster
Release the Reader Node when finished

6. Wire an indicator to the output "Timeout" to know when the timeout has expired without reading valid data.
7. Enable "Only New Samples" to remain blocked until new samples arrive.
8. Run the writer.

9. Notice that samples are received, the Timeout indicator is off, and the loop iteration counter increases rapidly (depending on the publication speed).
10. Stop the writer.
11. Notice that the Timeout indicator is on and the loop iteration counter increases once every second.
12. Switch the “Read Mode” to LVDDS_READ_MODE_POLLING and notice that the loop counter increases rapidly again.



Overview: Demonstrate how to subscribe to and read a Cluster type using Connext DDS.
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2 .Check that "Read Mode" is set to LVDDS_READ_MODE_BLOCKING_POLLING .
3. Enable "Only New Samples" to only show new samples.
4. Run the **RTI Connext DDS Cluster Writer.vi** example.
5. Observe how the "Loop Iteration" increases continuously and the "Timeout" indicator remains off.
6. Increase the seconds (sec) in the "Blocking Timeout" control to two.
7. Change "Read Mode" to LVDDS_READ_MODE_BLOCKING_READ.
8. Stop the writer and see how the "Iteration Counter" now increases once per two seconds and the timeout is on.
9. Run the writer again. "Timeout" should be off and "Loop Iteration" should increase continuously again.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

DDS Sample Info
DDS_SampleStateKind: DDS_INVALID_SAMPLE_STATE
DDS_ViewStateKind: DDS_INVALID_SAMPLE_STATE
DDS_InstanceStateKind: DDS_INVALID_STATE
Source Timestamp
sec: 0
nanosec: 0
valid_data

Query condition:
Only New Samples
Loop Iteration: 533
Timeout
Read Mode: LVDDS_READ_MODE_BLOCKING
Blocking Timeout
sec: 1
nanosec: 0

complexType
Text:
I32_Num: 0
I64_Num: 0
U16_Num: 0
Sgl_Array: 0 | 1, 1, 1, 1
innercluster
Sgl_Num: 0
Boolean
superinnerCluster
Dbl_Num: 0
ultrainnerCluster
I16_Array: 0 | 0, 0

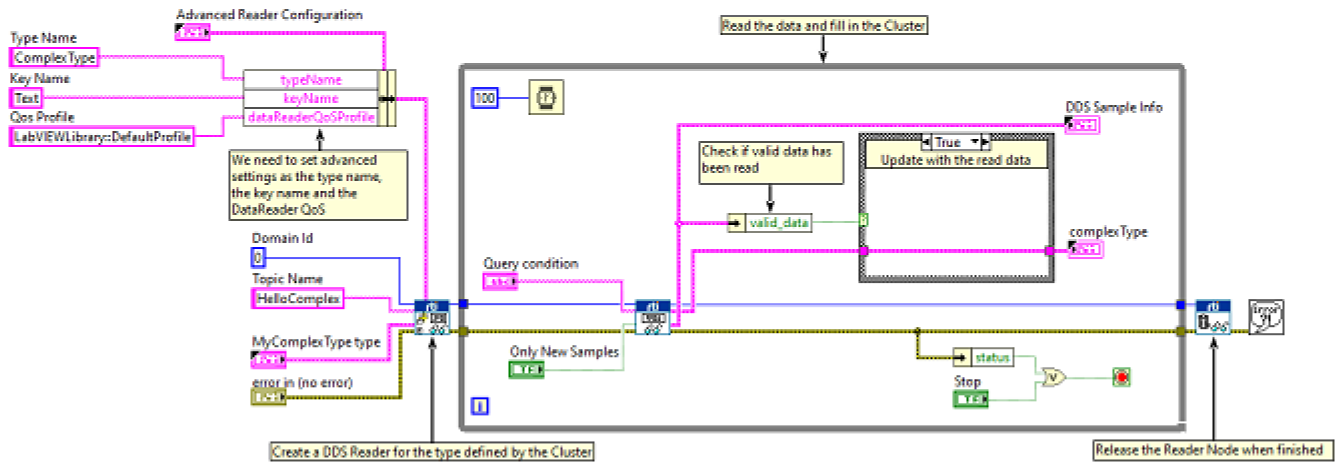
4.4 Lesson 4–Filtering Data

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 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38.

4.4.1 Filtering Data Using Query Conditions

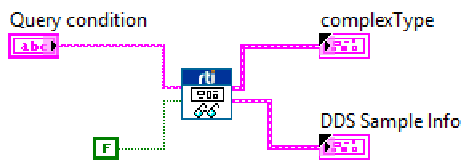
1. Open the **Tutorial_Cluster Reader Example** from 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38 and save it as a new VI named **Tutorial_Filter_Cluster.vi**. The Block Diagram should look similar to this:


Advanced Reader Configuration
Type Name
ComplexType
Key Name
Text
Qos Profile
LabVIEWLibrary::DefaultProfile
typeName
keyName
dataReaderQoSProfile
We need to set advanced settings as the type name, the key name and the DataReader QoS
Domain Id
0
Topic Name
HelloComplex
MyComplexType type
error in (no error)
Create a DDS Reader for the type defined by the Cluster
Read the data and fill in the Cluster
100
Query condition
Only New Samples
Check if valid data has been read
valid_data
True
Update with the read data
DDS Sample Info
complexType
status
Stop
Release the Reader Node when finished

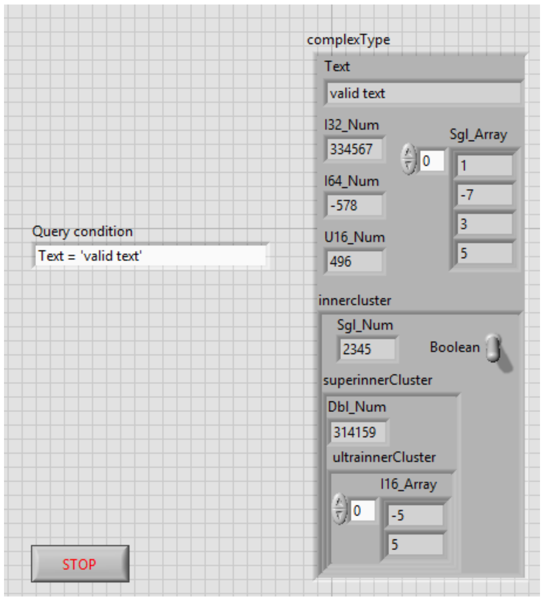
   With DDS, you can filter network data by subscribing to only the Topics of interest. Additionally, DDS provides the capability to filter data within a Topic by specifying a query condition for the data to match. The syntax of this Query Condition is similar to standard SQL queries. We will demonstrate how to filter data with various query conditions.

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_Filter_Cluster.vi**.



Query condition: Text = 'valid text'

complexType
Text: valid text
I32_Num: 334567
I64_Num: -578
U16_Num: 496
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 2345
superinnerCluster
Dbl_Num: 314159
ultrainnerCluster
I16_Array: 0 | -5, 5

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.

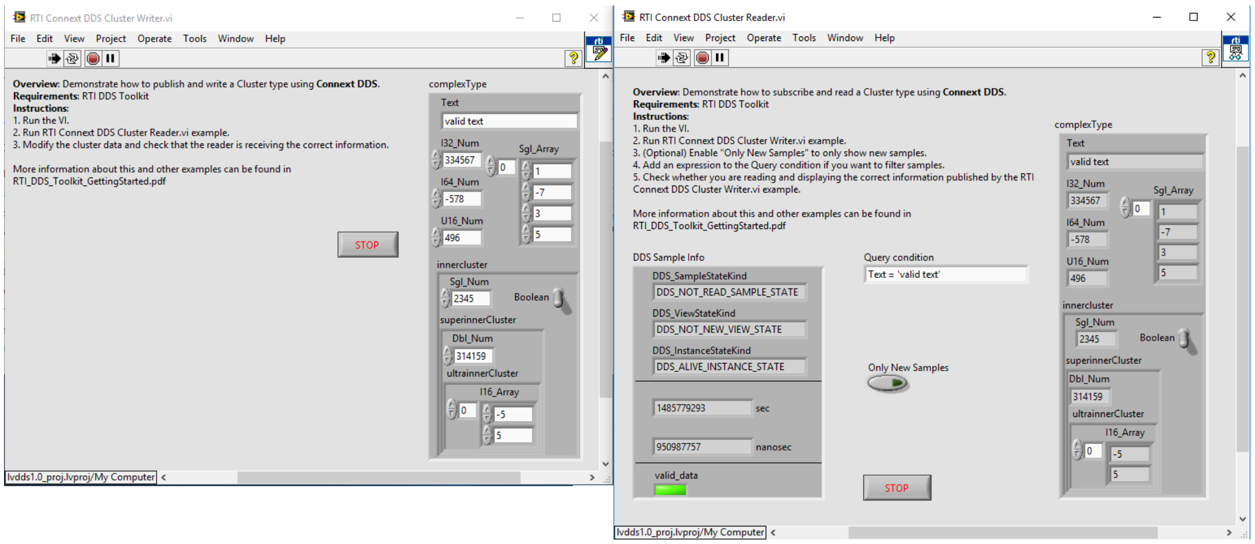
We will now filter data by content; for example, only read those samples where the cluster field is equal to "valid text":

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.


RTI Connext DDS Cluster Writer.vi
File Edit View Project Operate Tools Window Help
STOP
lvdds1.0_proj.lvproj/My Computer
RTI Connext DDS Cluster Reader.vi
File Edit View Project Operate Tools Window Help
STOP
lvdds1.0_proj.lvproj/My Computer

RTI Connext DDS Cluster Writer.vi

Overview: Demonstrate how to publish and write a Cluster type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Cluster Reader.vi example.
3. Modify the cluster data and check that the reader is receiving the correct information.

More information about this and other examples can be found in
RTI_DDS_Toolkit_GettingStarted.pdf

complexType
Text: valid text
I32_Num: 334567
I64_Num: -578
U16_Num: 496
Sgl_Array: 0; 1, -7, 3, 5
innercluster
Sgl_Num: 2345
Boolean
superinnerCluster
Dbl_Num: 314159
ultrainnerCluster
I16_Array: 0; -5, 5

RTI Connext DDS Cluster Reader.vi

Overview: Demonstrate how to subscribe and read a Cluster type using **Connext DDS.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Cluster Writer.vi example.
3. (Optional) Enable "Only New Samples" to only show new samples.
4. Add an expression to the Query condition if you want to filter samples.
5. Check whether you are reading and displaying the correct information published by the RTI Connext DDS Cluster Writer.vi example.

More information about this and other examples can be found in
RTI_DDS_Toolkit_GettingStarted.pdf

DDS Sample Info
DDS_SampleStateKind: DDS_NOT_READ_SAMPLE_STATE
DDS_ViewStateKind: DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind: DDS_ALIVE_INSTANCE_STATE
1485779293 sec
950987757 nanosec
valid_data

Query condition: Text = 'valid text'

Only New Samples

complexType
Text: valid text
I32_Num: 334567
I64_Num: -578
U16_Num: 496
Sgl_Array: 0; 1, -7, 3, 5
innercluster
Sgl_Num: 2345
Boolean
superinnerCluster
Dbl_Num: 314159
ultrainnerCluster
I16_Array: 0; -5, 5

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’”

4.4.2 Filtering Data Using ContentFilteredTopics

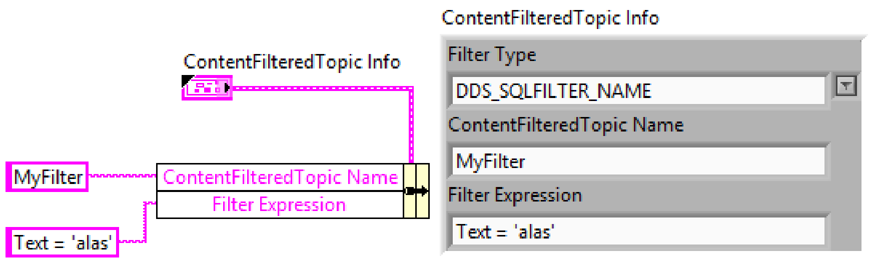
In this section we will learn how to use ContentFilteredTopics. This allows us to filter the data on the publisher side. See 6.9 Advanced Filtering of Data—ContentFilteredTopics on page 127 for further details.

This lesson assumes you have successfully completed the previous lesson in 4.4.1 Filtering Data Using Query Conditions on page 47.

1. Open **Tutorial_Read_Cluster.vi** from 4.4.1 Filtering Data Using Query Conditions on page 47 and save it as a new VI named **Tutorial_Content_Filter_Cluster.vi**.

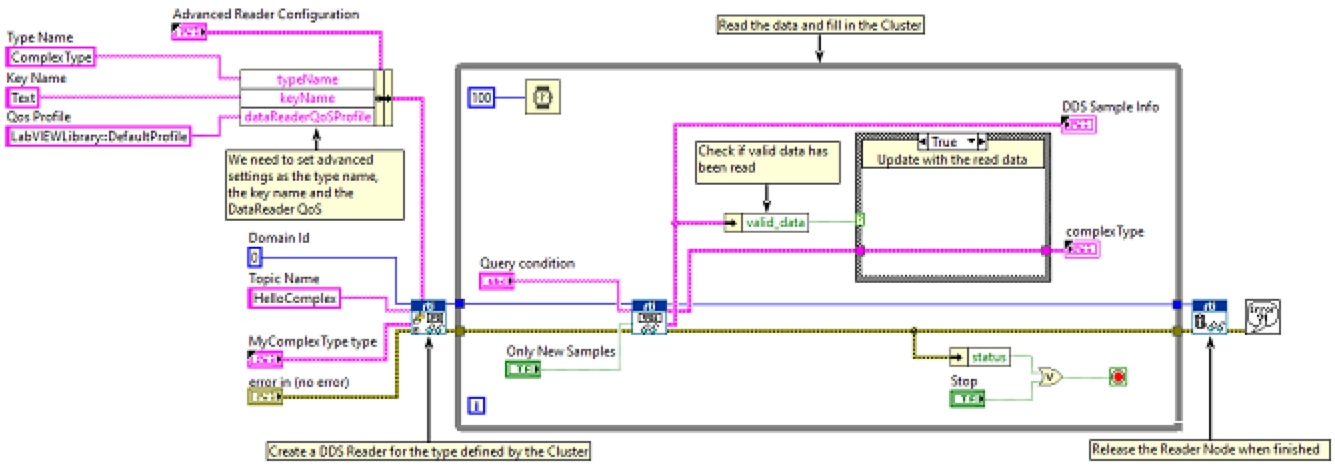
   With DDS, you can filter network data by subscribing to only the Topics of interest. Additionally, DDS provides the capability to filter data within a Topic by using a ContentFilteredTopic. A ContentFilteredTopic will not only makes possible to subscribe to topics but also specify that you are only interested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

2. Add a ContentFilteredTopic (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
Filter Type
DDS_SQLFILTER_NAME
ContentFilteredTopic Name
MyFilter
Filter Expression
Text = 'alas'

   Note: For details on Advanced Settings, see 6.5 Configuring Advanced Reader Settings on page 104.
   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:



We need to set advanced settings as the type name, the key name and the DataReader QoS

Read the data and fill in the Cluster

Create a DDS Reader for the type defined by the Cluster

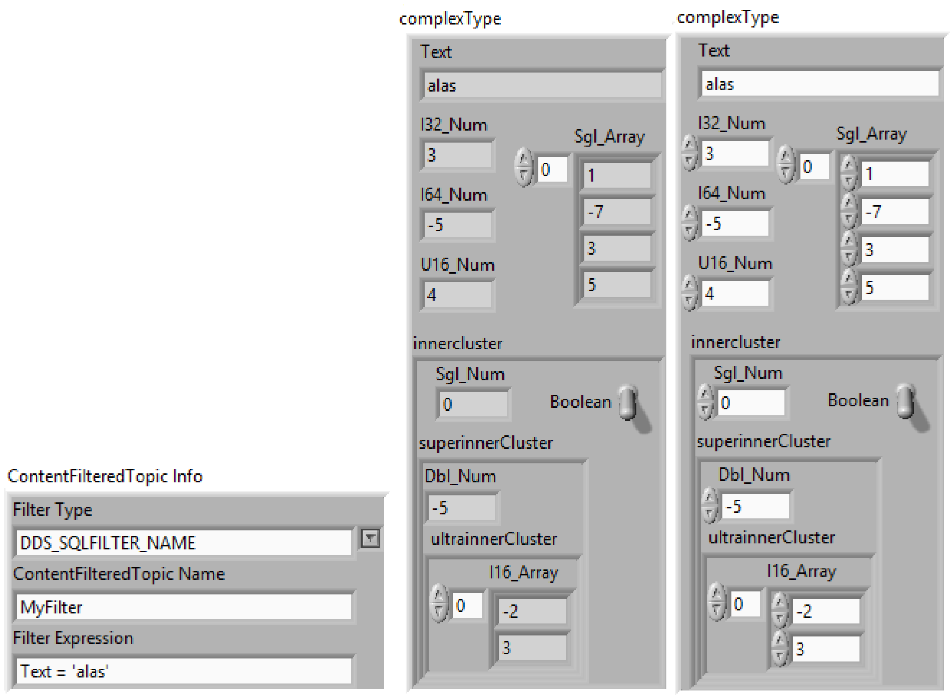
Release the Reader Node when finished

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).



ContentFilteredTopic Info
Filter Type: DDS_SQLFILTER_NAME
ContentFilteredTopic Name: MyFilter
Filter Expression: Text = 'alas'

complexType
Text: alas
I32_Num: 3
I64_Num: -5
U16_Num: 4
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 0
Boolean
superinnerCluster
Dbl_Num: -5
ultrainnerCluster
I16_Array: 0 | -2, 3

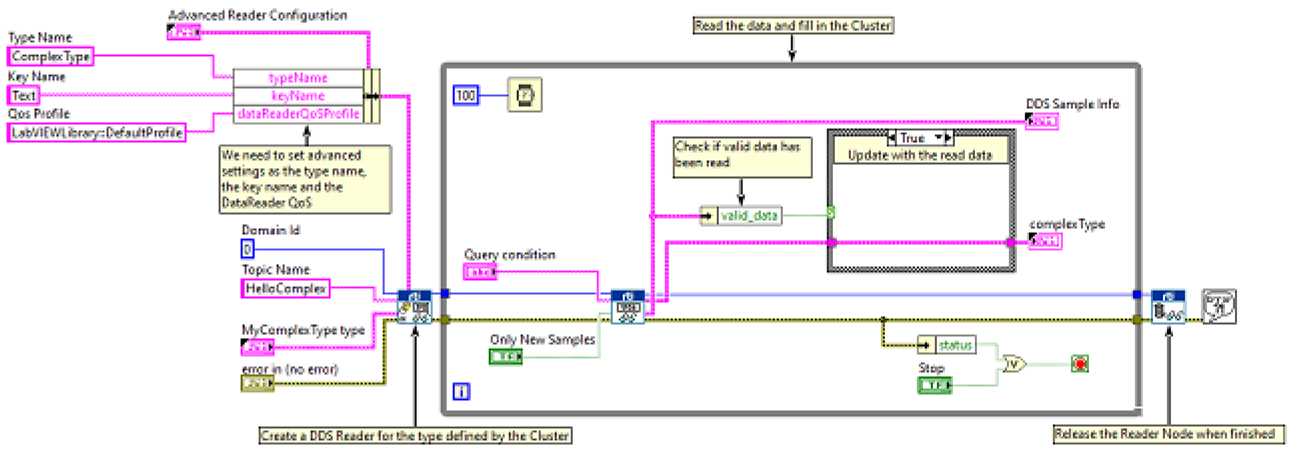
complexType
Text: alas
I32_Num: 3
I64_Num: -5
U16_Num: 4
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 0
Boolean
superinnerCluster
Dbl_Num: -5
ultrainnerCluster
I16_Array: 0 | -2, 3

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’”

4.5 Lesson 5–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 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38.

1. Open **Tutorial_Cluster Reader Example.vi** from 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38 and save as a new VI with the name **Tutorial_Only_New_Read.vi**. The Block Diagram should look similar to this:

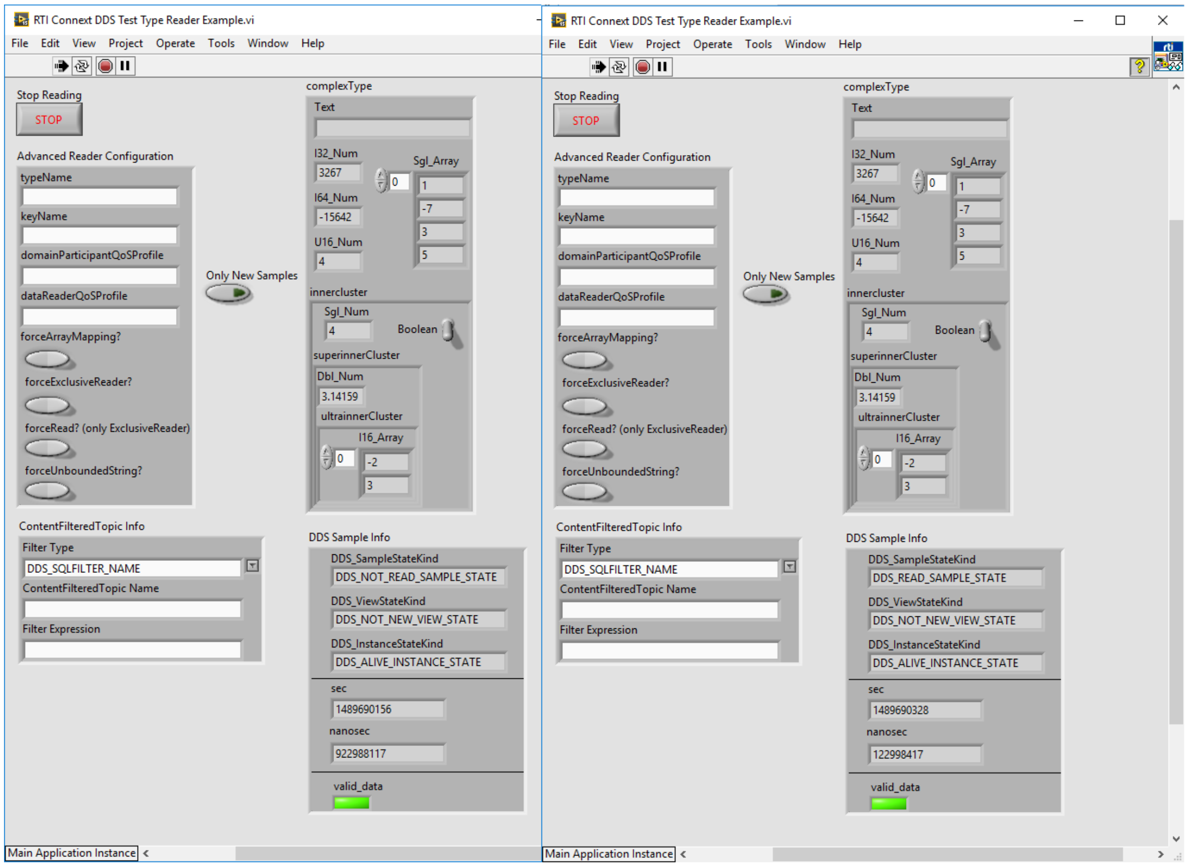

Advanced Reader Configuration
Type Name
ComplexType
Key Name
Text
Qos Profile
LabVIEWLibrary::DefaultProfile
typeName
keyName
dataReaderQoSProfile
We need to set advanced settings as the type name, the key name and the DataReader QoS
Domain Id
0
Topic Name
HelloComplex
MyComplexType type
error in (no error)
Create a DDS Reader for the type defined by the Cluster
Read the data and fill in the Cluster
100
DDS Sample Info
Check if valid data has been read
True
Update with the read data
valid_data
complexType
Query condition
Only New Samples
status
Stop
Release the Reader Node when finished

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 Sample*'s 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**.


RTI Connext DDS Test Type Reader Example.vi
File Edit View Project Operate Tools Window Help
Main Application Instance

RTI Connext DDS Test Type Reader Example.vi (left)

Stop Reading: STOP

Advanced Reader Configuration: typeName; keyName; domainParticipantQoSProfile; dataReaderQoSProfile; forceArrayMapping?; forceExclusiveReader?; forceRead? (only ExclusiveReader); forceUnboundedString?

Only New Samples

complexType: Text; I32_Num 3267; I64_Num -15642; U16_Num 4; Sgl_Array 0: 1, -7, 3, 5; innercluster: Sgl_Num 4, Boolean; superinnerCluster: Dbl_Num 3.14159; ultrainnerCluster: I16_Array 0: -2, 3

ContentFilteredTopic Info: Filter Type DDS_SQLFILTER_NAME; ContentFilteredTopic Name; Filter Expression

DDS Sample Info: DDS_SampleStateKind DDS_NOT_READ_SAMPLE_STATE; DDS_ViewStateKind DDS_NOT_NEW_VIEW_STATE; DDS_InstanceStateKind DDS_ALIVE_INSTANCE_STATE; sec 1489690156; nanosec 922988117; valid_data

RTI Connext DDS Test Type Reader Example.vi (right)

Stop Reading: STOP

Advanced Reader Configuration: typeName; keyName; domainParticipantQoSProfile; dataReaderQoSProfile; forceArrayMapping?; forceExclusiveReader?; forceRead? (only ExclusiveReader); forceUnboundedString?

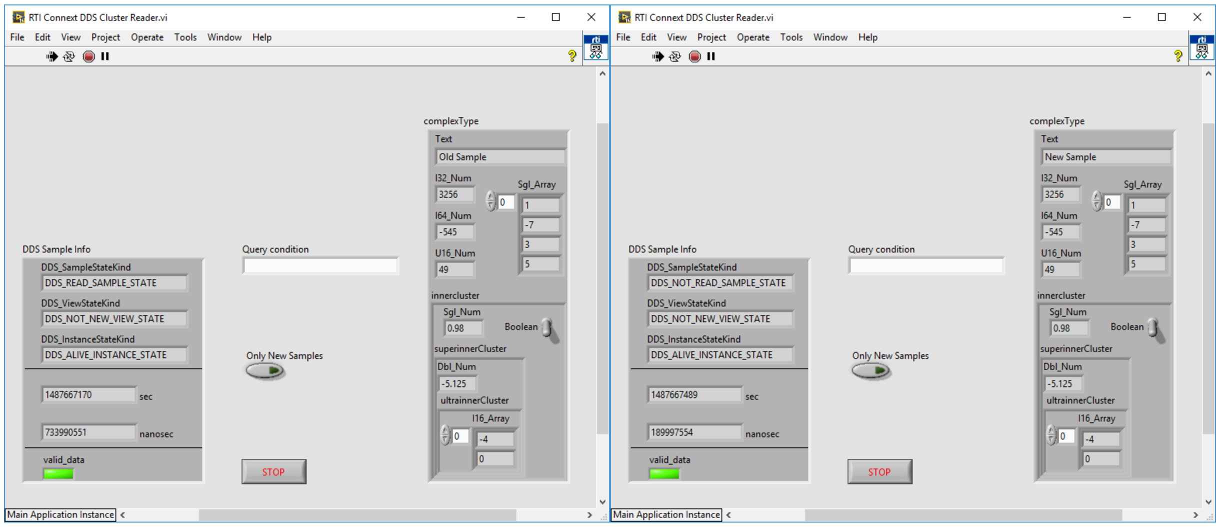
Only New Samples

complexType: Text; I32_Num 3267; I64_Num -15642; U16_Num 4; Sgl_Array 0: 1, -7, 3, 5; innercluster: Sgl_Num 4, Boolean; superinnerCluster: Dbl_Num 3.14159; ultrainnerCluster: I16_Array 0: -2, 3

ContentFilteredTopic Info: Filter Type DDS_SQLFILTER_NAME; ContentFilteredTopic Name; Filter Expression

DDS Sample Info: DDS_SampleStateKind DDS_READ_SAMPLE_STATE; DDS_ViewStateKind DDS_NOT_NEW_VIEW_STATE; DDS_InstanceStateKind DDS_ALIVE_INSTANCE_STATE; sec 1489690328; nanosec 122998417; valid_data

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.


RTI Connext DDS Cluster Reader.vi
File Edit View Project Operate Tools Window Help
STOP
Main Application Instance

RTI Connext DDS Cluster Reader.vi (left window)

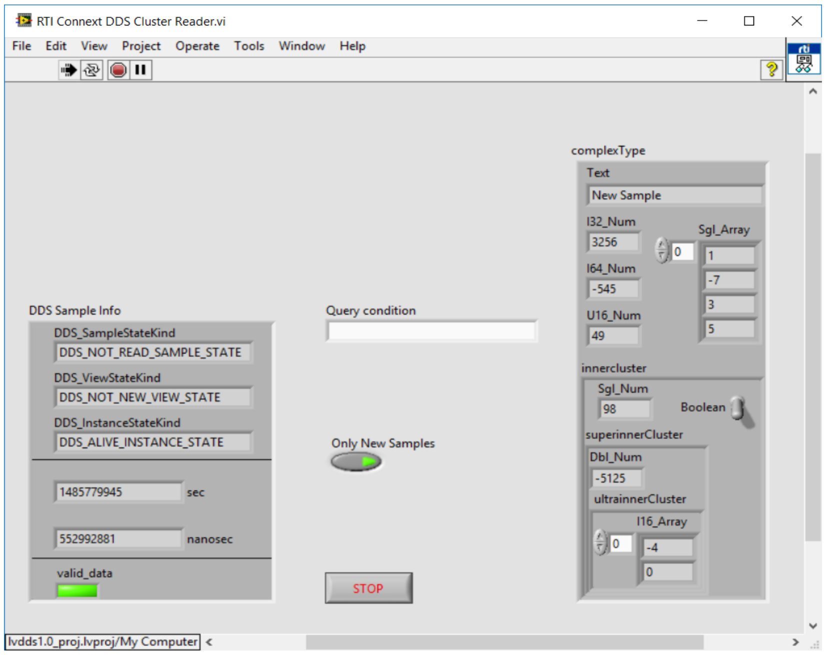
DDS Sample Info
DDS_SampleStateKind: DDS_READ_SAMPLE_STATE
DDS_ViewStateKind: DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind: DDS_ALIVE_INSTANCE_STATE
1487667170 sec
733990551 nanosec
valid_data
Query condition:
Only New Samples
complexType
Text: Old Sample
I32_Num: 3256
I64_Num: -545
U16_Num: 49
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 0.98
Boolean
superinnerCluster
Dbl_Num: -5.125
ultrainnerCluster
I16_Array: 0 | -4, 0

RTI Connext DDS Cluster Reader.vi (right window)

DDS Sample Info
DDS_SampleStateKind: DDS_NOT_READ_SAMPLE_STATE
DDS_ViewStateKind: DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind: DDS_ALIVE_INSTANCE_STATE
1487667489 sec
189997554 nanosec
valid_data
Query condition:
Only New Samples
complexType
Text: New Sample
I32_Num: 3256
I64_Num: -545
U16_Num: 49
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 0.98
Boolean
superinnerCluster
Dbl_Num: -5.125
ultrainnerCluster
I16_Array: 0 | -4, 0

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 4.8 Lesson 8—Reading All Samples (Reliable Communication) on page 64).

   Note: A different approach is to use Exclusive Readers and 'take' to guarantee that the data will only be read once (see 6.1 Default Configuration: DDS Entities Created by 'Simple Create' SubVIs on page 97 and 4.8.2 Writing and Reading using Strict Reliability on page 68).


RTI Connext DDS Cluster Reader.vi
File Edit View Project Operate Tools Window Help
STOP
lvdds1.0_proj.lvproj/My Computer

DDS Sample Info
DDS_SampleStateKind: DDS_NOT_READ_SAMPLE_STATE
DDS_ViewStateKind: DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind: DDS_ALIVE_INSTANCE_STATE
1485779945 sec
552992881 nanosec
valid_data
Query condition:
Only New Samples
complexType
Text: New Sample
I32_Num: 3256
I64_Num: -545
U16_Num: 49
Sgl_Array: 0 | 1, -7, 3, 5
innercluster
Sgl_Num: 98
Boolean
superinnerCluster
Dbl_Num: -5125
ultrainnerCluster
I16_Array: 0 | -4, 0

4.6 Lesson 6–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 https://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).

4.6.1 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)

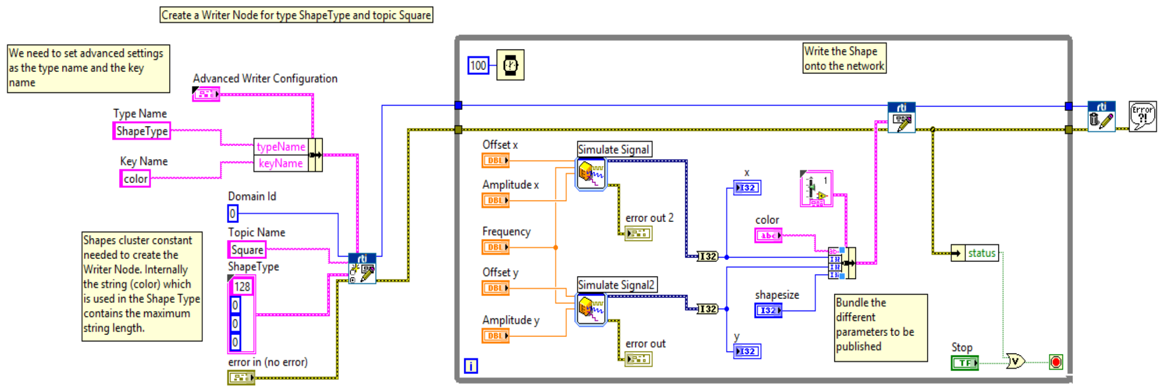
Shapes Demo can publish three different Topics of type ShapeType:

- Square
- Circle
- Triangle

4.6.2 Publishing a Shape (Square)

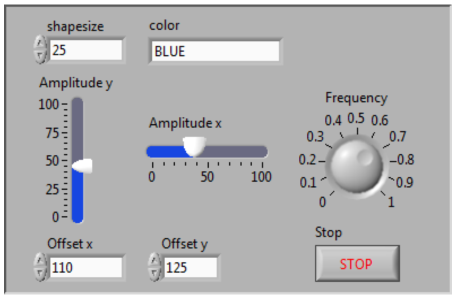
We will use LabVIEW to publish a square in domain 0. Additionally, we will generate two sine functions for the ShapeType X and Y coordinates in order to move the square in a circular or elliptical pattern.

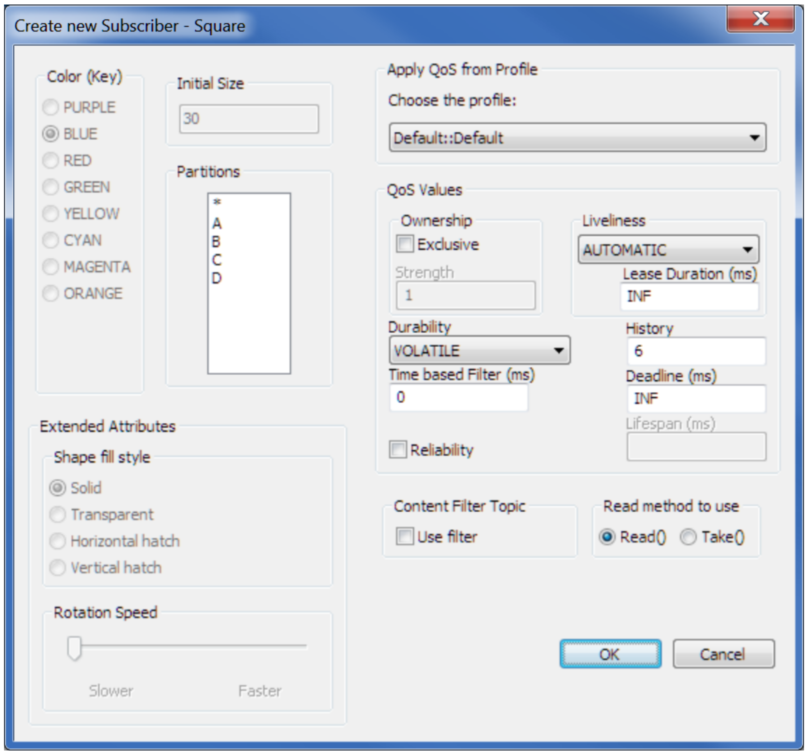
1. Open **RTI Connext DDS Shapes Writer.vi** from the LabVIEW examples **ShapesDemo** directory under **RTI DDS Toolkit**. (Instructions for finding the examples are in 1.6 LabVIEW Examples on page 12.)
2. Open the Block Diagram and note that the VI is creating a *Writer* object to publish a ShapeType 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.


Create a Writer Node for type ShapeType and topic Square
We need to set advanced settings as the type name and the key name
Advanced Writer Configuration
Type Name
ShapeType
typeName
Key Name
color
keyName
Domain Id
0
Topic Name
Square
ShapeType
128
0
0
0
Shapes cluster constant needed to create the Writer Node. Internally the string (color) which is used in the Shape Type contains the maximum string length.
error in (no error)
100
Write the Shape onto the network
Offset x
Amplitude x
Frequency
Offset y
Amplitude y
Simulate Signal
Simulate Signal2
error out 2
error out
x
y
color
shapesize
Bundle the different parameters to be published
status
Stop

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**.


shapesize: 25
color: BLUE
Offset x: 110
Offset y: 125


Create new Subscriber - Square

Color (Key): PURPLE [✓ BLUE] RED GREEN YELLOW CYAN MAGENTA ORANGE

Initial Size: 30

Partitions: * A B C D

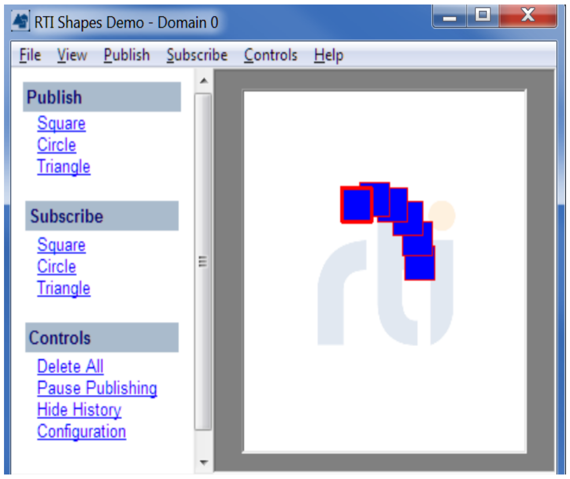
Apply QoS from Profile
Choose the profile: Default::Default

QoS Values
Ownership: Exclusive ☐
Strength: 1
Liveliness: AUTOMATIC
Lease Duration (ms): INF
Durability: VOLATILE
History: 6
Time based Filter (ms): 0
Deadline (ms): INF
Lifespan (ms):
Reliability ☐

Extended Attributes
Shape fill style: [✓ Solid] Transparent Horizontal hatch Vertical hatch
Rotation Speed: Slower — Faster

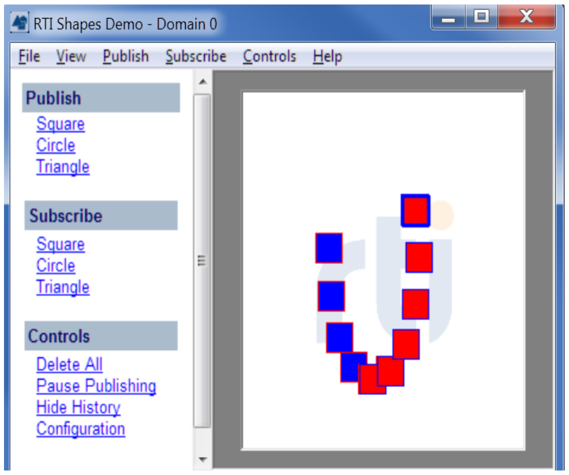
Content Filter Topic: Use filter ☐
Read method to use: [✓ Read()] Take()

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.

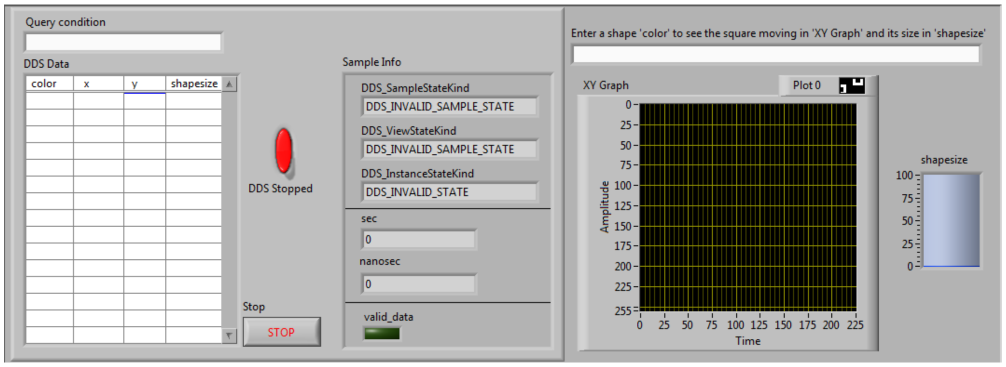


Note: When you change the square's color, you will still see the blue square. This is because we defined Square as the Topic and Color as the Topic Key (instance). Using Keys allows the definition of a single Topic with multiple instances. When you change the color, you are publishing a new instance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

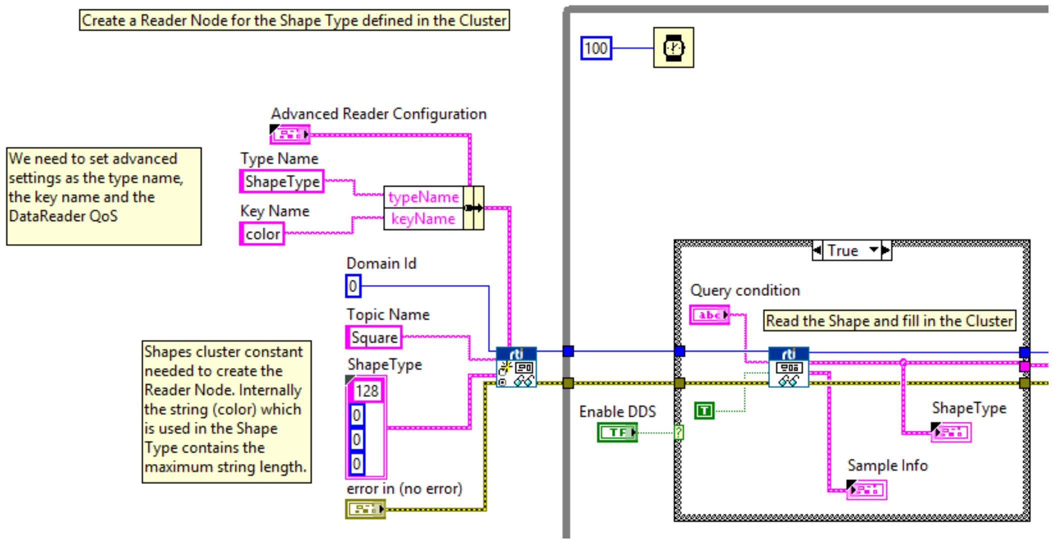
4.6.3 Subscribing to Shapes

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 4.4 Lesson 4—Filtering Data on page 47. Finally, we have the Stop button that stops the whole VI.
   - 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**.



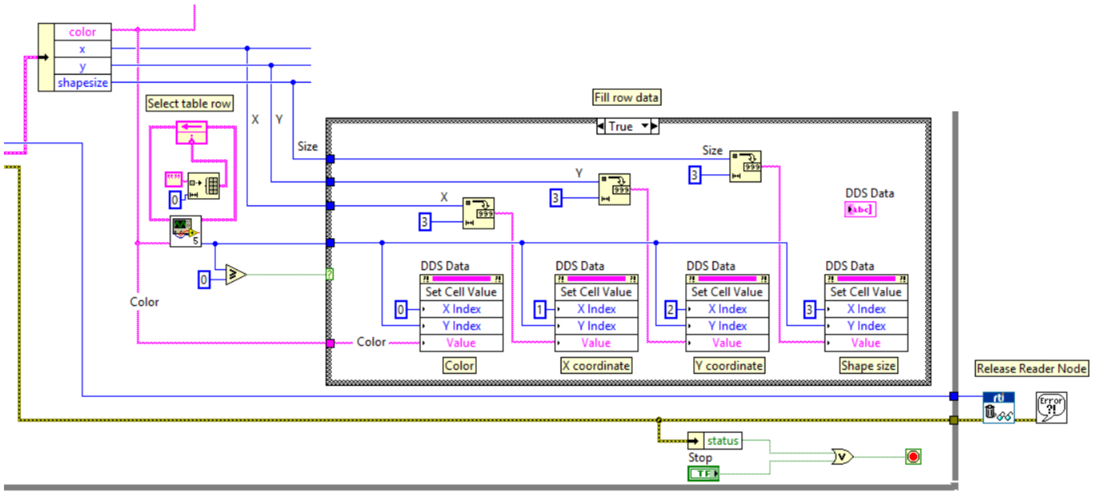
3. Open the Block Diagram and review the three different processes:
   a. Creating the *Reader* object and reading:
      - A *Reader* object is created to subscribe to the type **ShapeType** and the Topic **Square**, also providing a correct **ShapesDemo** cluster in the **data type** pin.
      - Once 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.


Create a Reader Node for the Shape Type defined in the Cluster
100
Advanced Reader Configuration
We need to set advanced settings as the type name, the key name and the DataReader QoS
Type Name
ShapeType
Key Name
color
typeName
keyName
Domain Id
0
Topic Name
Square
ShapeType
128
0
0
0
Shapes cluster constant needed to create the Reader Node. Internally the string (color) which is used in the Shape Type contains the maximum string length.
error in (no error)
Enable DDS
True
Query condition
Read the Shape and fill in the Cluster
ShapeType
Sample Info

   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.

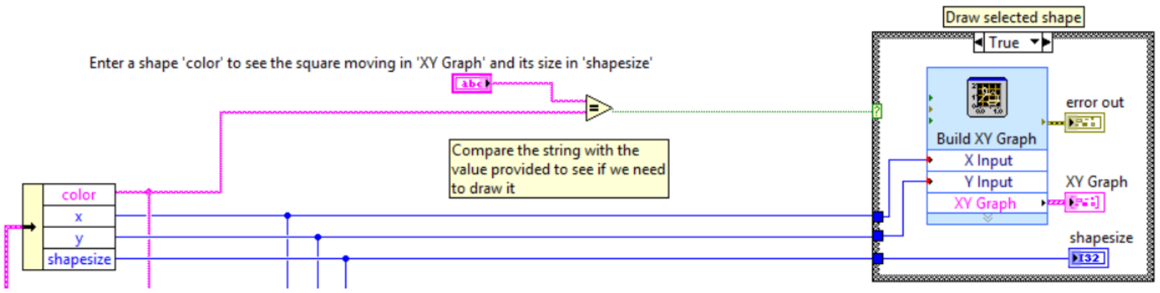


Note: This example uses a string with a max length of 128. To the maximum length, we have included the max size as the text value in the String in the input of the creation subVI.

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**.

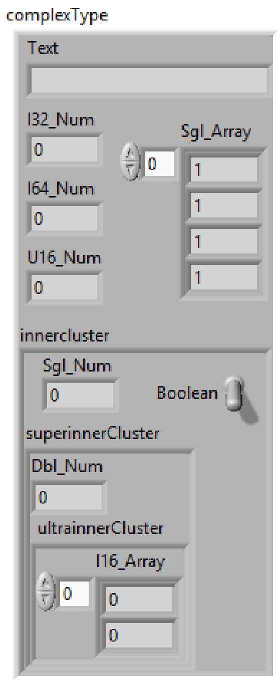

Draw selected shape
True
Enter a shape 'color' to see the square moving in 'XY Graph' and its size in 'shapesize'
Compare the string with the value provided to see if we need to draw it
error out
Build XY Graph
X Input
Y Input
XY Graph
XY Graph
shapesize
color
x
y
shapesize

4.7 Lesson 7–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 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38. You can also use the provided example, **RTI Connext DDS Cluster Example Reader/Writer.vi**.

4.7.1 Adding Multiple Top-Level Fields as Keys

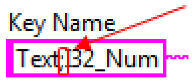
1. Open **Tutorial_Cluster Reader Example.vi** from 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38 and save it as a new VI named **Tutorial_MultipleKey_Read_Cluster.vi**.



   As you can see in the figure to the right, our cluster is quite complex and includes many fields.

   In 4.6 Lesson 6—Using Keyed Types (RTI Shapes Demo) on page 55, 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:

```
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
    long long I64_Num;      //@ID 2
    unsigned short U16_Num; //@ID 3
    sequence<float,4> Sgl_Array;    //@ID 4
    innerclusterType innercluster;  //@ID 5
};
//@Extensibility EXTENSIBLE_EXTENSIBILITY
```

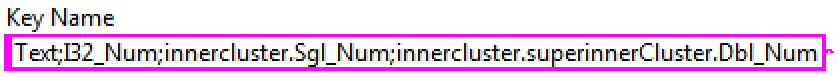
Note: The key name specification is case sensitive.

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

4.7.2 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 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38 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;I32_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:

```
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.

4.8 Lesson 8–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 6.1 Default Configuration: DDS Entities Created by 'Simple Create' SubVIs on page 97) or using exclusive Reader nodes.

The first approach is explained in 4.8.1 Writing and Reading Reliably Using the Default Configuration below. The latter approach is explained in 4.8.2 Writing and Reading using Strict Reliability on page 68.

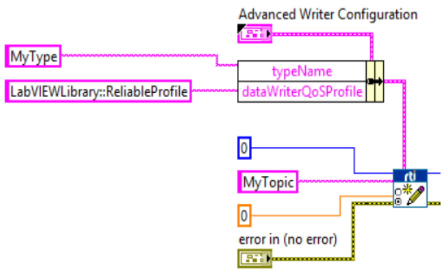
4.8.1 Writing and Reading Reliably Using the Default Configuration

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

4.8.1.1 Writing Reliably

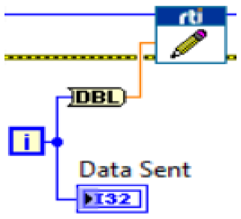
1. Open a blank VI and open the Block Diagram.

   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.

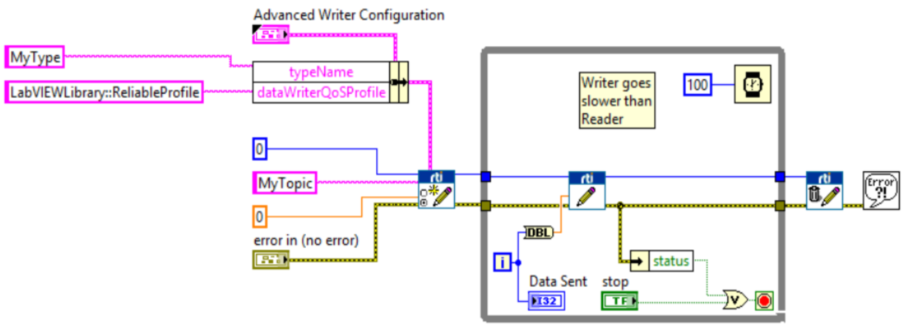


   For details on the *Advanced Create Writer* subVI, see Chapter 6 Advanced Concepts and Settings on page 96.

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**.



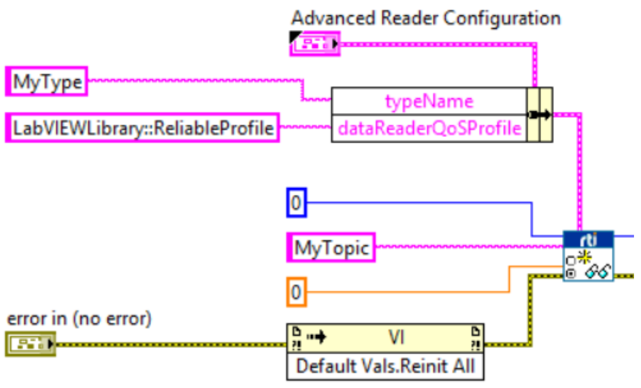
3. Add a *Release Writer* subVI and complete the VI as shown in the following figure. Pay special attention to the *Wait* function.


Advanced Writer Configuration
MyType
LabVIEWLibrary::ReliableProfile
typeName
dataWriterQoSProfile
0
MyTopic
0
error in (no error)
Writer goes slower than Reader
100
DBL
status
Data Sent
stop

4. Save it as **Tutorial_Write_Reliable.vi**.

4.8.1.2 Reading Reliably

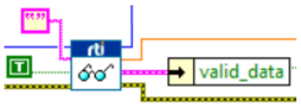
1. Open a blank VI and create an indicator of an array of doubles. Show the vertical scroll bar of the array in the array properties, i.e., right-click in the array, select **Properties** and check the **Show Vertical Scroll Bar** option.
2. Add an *Advanced Create Reader* subVI and fill in the parameters to create a *Reader* of doubles, as shown in the following figure. Pay close attention to the new QoS Profile.



For details on the *Advanced Create Reader* subVI, see Chapter 6 Advanced Concepts and Settings on page 96.

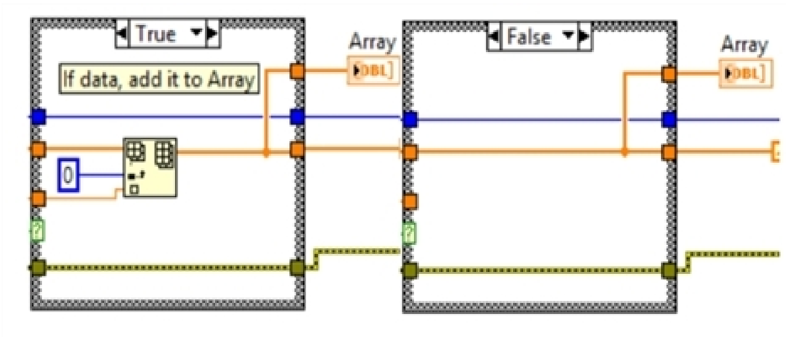
3. Optionally, add an **Invoke note** to call the method **Reinitialize All to Default**. This function resets all the controls and indicators in the VI to the default value. To include it, follow this steps:
   a. Find **Invoke Node** under **Programming, Application Control.**
   b. Right-click on 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)**.



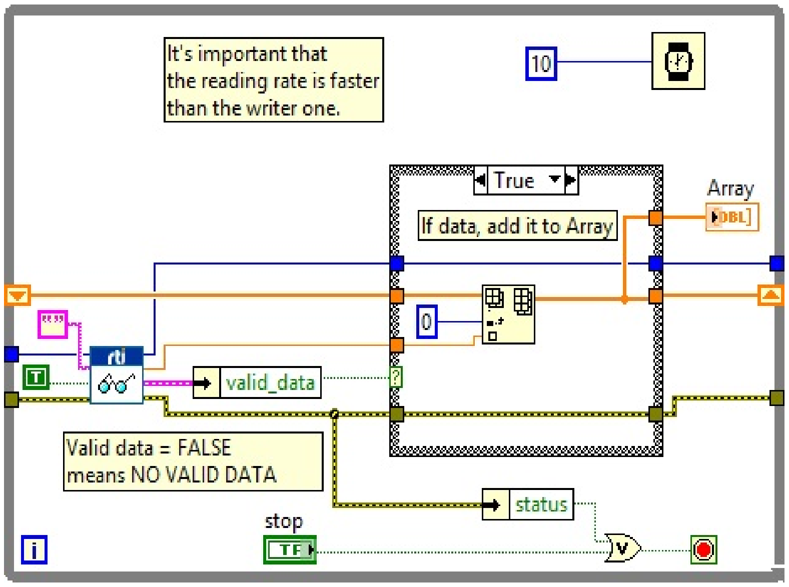
    For details on the *Read* subVI, see A.2.2 Reader on page 137.

5. If the data is valid, insert it in the array. Otherwise, ignore the data:

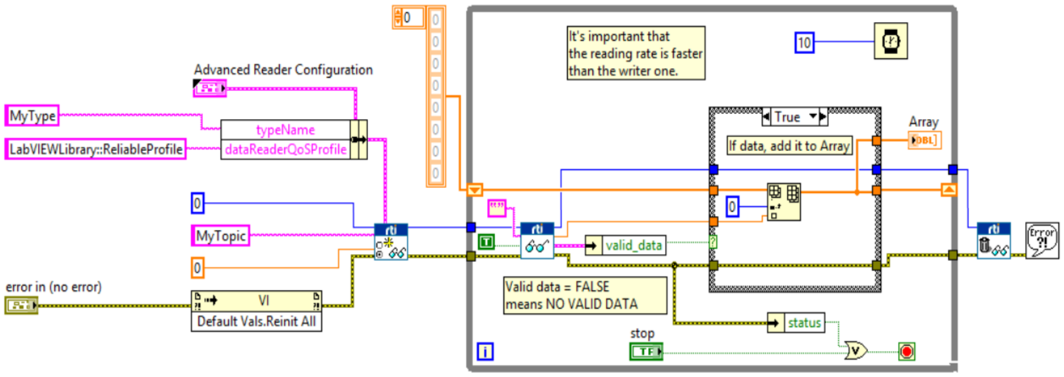


    a. Create a **Case Structure** from **Programming**, **Structures** and connect the output of **valid_data** to the question mark.
    b. Create an array indicator and connect it to the output of the **Case Structure**.
    c. Connect the *Read* subVI outputs as inputs of the **Case Structure**, except **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.

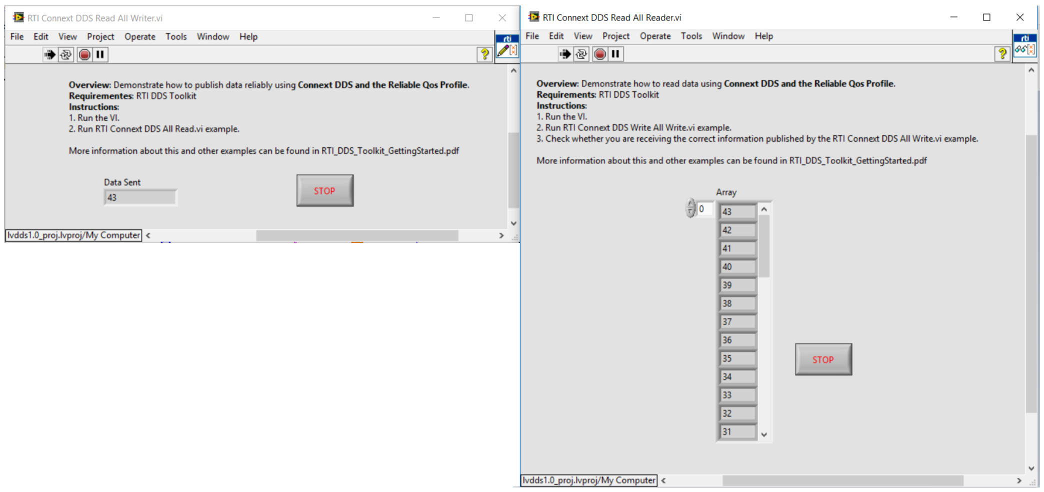

It's important that the reading rate is faster than the writer one.
10
True
If data, add it to Array
Array
DBL
0
rti
valid_data
Valid data = FALSE means NO VALID DATA
status
stop
TF

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 on page 94 for more information.


Advanced Reader Configuration
MyType
LabVIEWLibrary::ReliableProfile
typeName
dataReaderQoSProfile
0
MyTopic
0
error in (no error)
VI
Default Vals.Reinit All
It's important that the reading rate is faster than the writer one.
10
True
If data, add it to Array
Array
valid_data
Valid data = FALSE means NO VALID DATA
status
stop
Error

For details on the *Release Reader* subVI, see A.2.2 Reader on page 137.

8. Save it as **Tutorial_Read_Reliable.vi**.


RTI Connext DDS Read All Writer.vi
File Edit View Project Operate Tools Window Help

Overview: Demonstrate how to publish data reliably using **Connext DDS and the Reliable Qos Profile.**
Requirementes: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS All Read.vi example.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Data Sent: 43

STOP
lvdds1.0_proj.lvproj/My Computer
RTI Connext DDS Read All Reader.vi
File Edit View Project Operate Tools Window Help

Overview: Demonstrate how to read data using **Connext DDS and the Reliable Qos Profile.**
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run RTI Connext DDS Write All Write.vi example.
3. Check whether you are receiving the correct information published by the RTI Connext DDS All Write.vi example.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Array: 0 | 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31

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

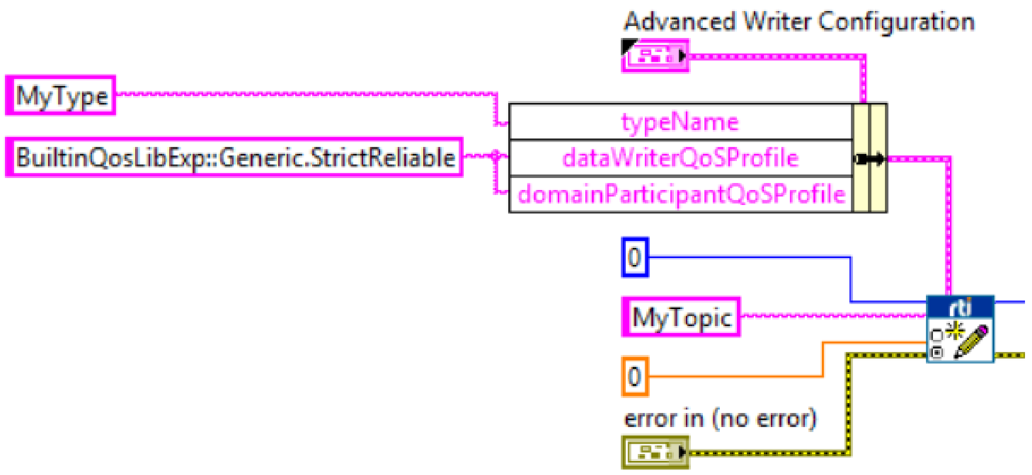
4.8.2 Writing and Reading using Strict Reliability

4.8.1 Writing and Reading Reliably Using the Default Configuration on page 64 assumes you are using the default configuration of *RTI DDS Toolkit*. As explained in Chapter 6 Advanced Concepts and Settings on page 96, 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* (for details on Built-in profiles, see the RTI Community Forum: http://-community.rti.com/examples/built-qos-profiles).

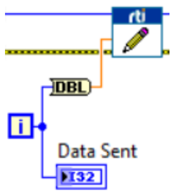
4.8.2.1 Writing in Strictly Reliable Mode

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:

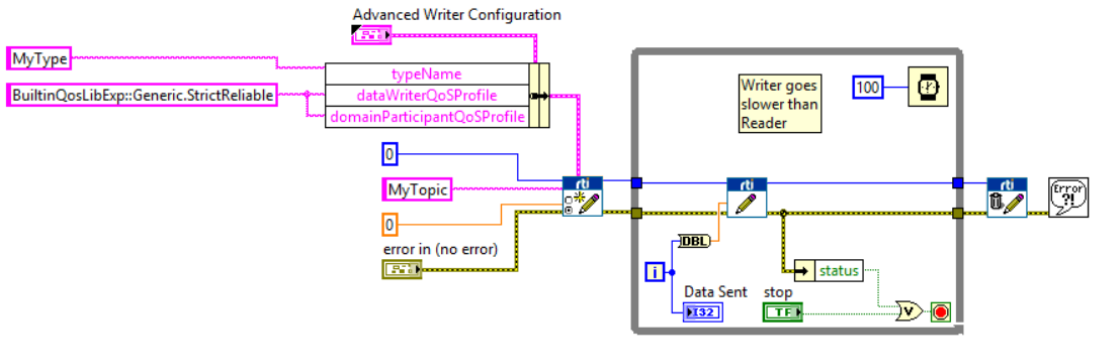

Advanced Writer Configuration
MyType
BuiltinQosLibExp::Generic.StrictReliable
typeName
dataWriterQoSProfile
domainParticipantQoSProfile
0
MyTopic
0
error in (no error)

For details on the *Create Advanced Writer* subVI, see Chapter 6 Advanced Concepts and Settings on page 96.

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.

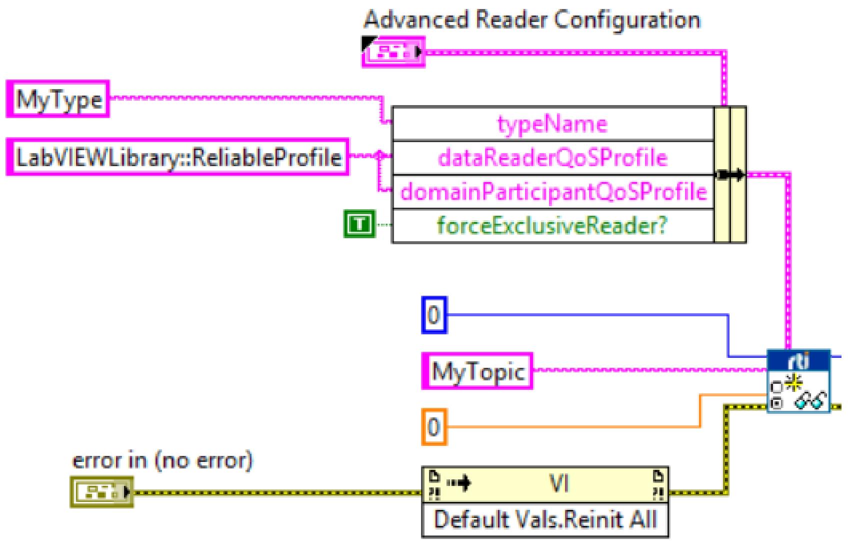

Advanced Writer Configuration
MyType
BuiltinQosLibExp::Generic.StrictReliable
typeName
dataWriterQoSProfile
domainParticipantQoSProfile
0
MyTopic
0
error in (no error)
Writer goes slower than Reader
100
DBL
Data Sent
I32
status
stop
TF

5. Save it as **Tutorial_Write_StrictReliable.vi**.

4.8.2.2 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.

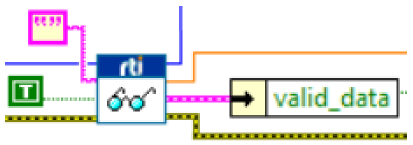
2. Add an *Advanced Create Reader* subVI and fill in the parameters to create a *Reader* of doubles, as shown in the following figure. Make sure you set the QoS profiles and the force**ExclusiveReader?** as shown in the following figure.



For details on the *Create Advanced Writer* subVI, see Chapter 6 Advanced Concepts and Settings on page 96.

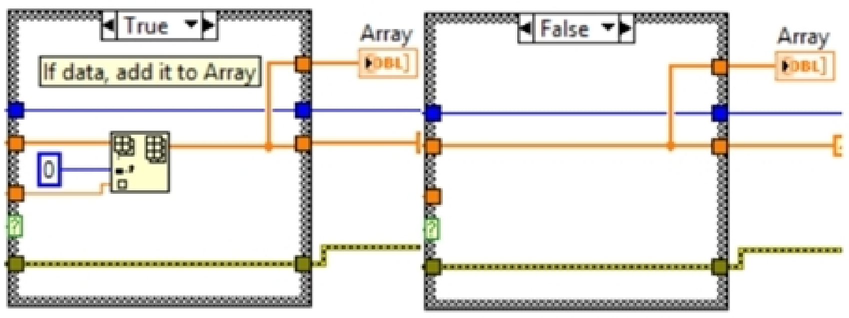
3. Optionally, add an *Invoke* note to call the method **Reinitialize All to Default**. This function resets all the controls and indicators in the VI to the default value. To include it, follow this steps:
   a. Find **Reinitialize All to Default** under **Programming, Application Control.**
   b. Right 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.

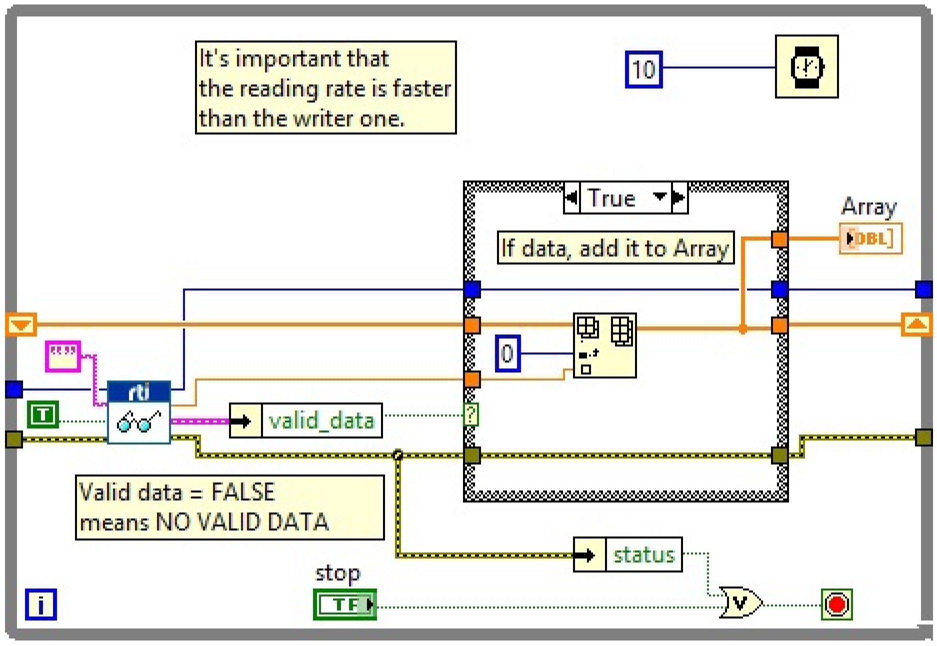


For details on the *Read* subVI, see A.2.2 Reader on page 137.

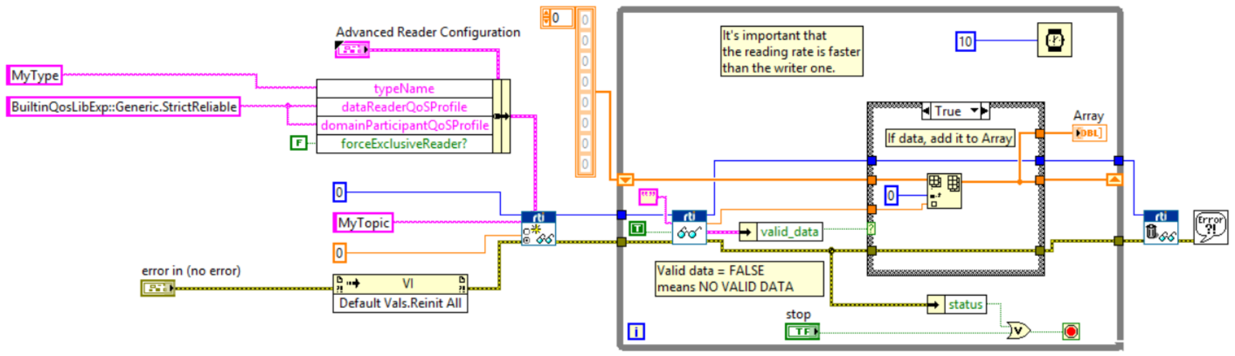
5. If the data is valid, insert it in the array. Otherwise, ignore the data:



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.


It's important that the reading rate is faster than the writer one.
10
True
If data, add it to Array
Array
DBL
0
rti
valid_data
Valid data = FALSE means NO VALID DATA
status
stop
TF
i

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.


Advanced Reader Configuration
MyType
BuiltinQosLibExp::Generic.StrictReliable
typeName
dataReaderQoSProfile
domainParticipantQoSProfile
forceExclusiveReader?
MyTopic
error in (no error)
VI
Default Vals.Reinit All
It's important that the reading rate is faster than the writer one.
10
True
If data, add it to Array
Array
valid_data
Valid data = FALSE means NO VALID DATA
status
stop

For details on the *Release Reader* subVI, see A.2.2 Reader on page 137.

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*.

4.9 Lesson 9–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[1]/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.

  4.9.1 Debugging an Application Using the Administration Panel below
- The monitoring profile, which enables *RTI Monitoring Library*.

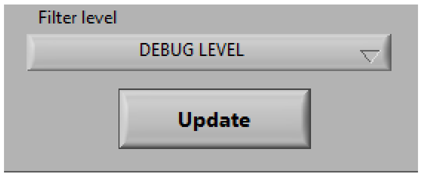
  4.9.2 Adapting a VI to Use RTI Monitoring Library on page 76

4.9.1 Debugging an Application Using the Administration Panel

Let's begin by opening the *Reader* and *Writer* VIs creation in 4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) on page 30. 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 6.7.1 Using Administration Panel (for Windows Systems only) on page 110.

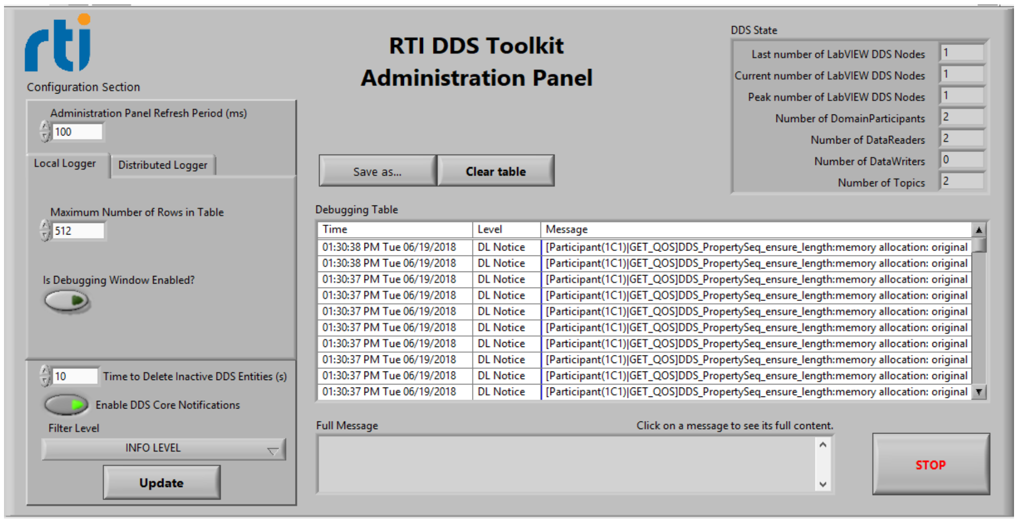
   Note: The Administration Panel may not work on RT Targets. If you want to read messages from a RT Target, you can deploy the VI described in 6.7.2.6 Reading Logged Messages on page 118.
2. Run the VI.
3. Set the Filter Level to be **DEBUG LEVEL**. This will cause all messages with log level of Debug 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 4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) on page 30 and click **Run**.
6. Go back to the Administration Panel. You will see the generated debugging messages in the

[1]On 64-bit systems, the folder is "Program Files (x86)"

Debugging table:



Administration Panel Refresh Period (ms): 100

Maximum Number of Rows in Table: 512

Is Debugging Window Enabled?

10 Time to Delete Inactive DDS Entities (s)

Enable DDS Core Notifications

Filter Level: INFO LEVEL

DDS State

Last number of LabVIEW DDS Nodes	1
Current number of LabVIEW DDS Nodes	1
Peak number of LabVIEW DDS Nodes	1
Number of DomainParticipants	2
Number of DataReaders	2
Number of DataWriters	0
Number of Topics	2

Debugging Table

Time	Level	Message
01:30:38 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:38 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original

Full Message

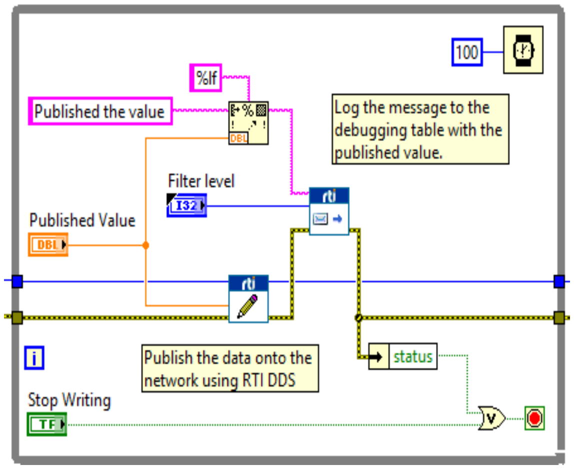
Click on a message to see its full content.

4.9.1.1 Logging Messages Manually

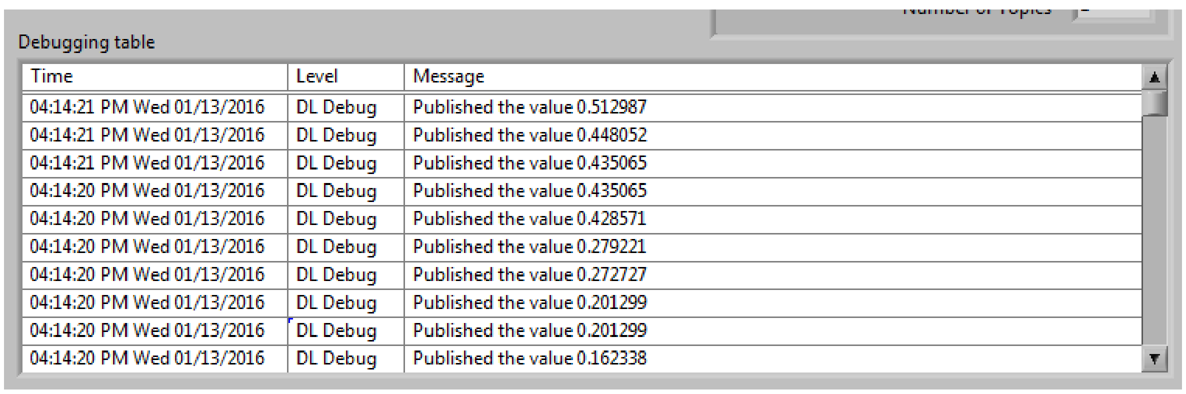
Now that we can debug our application, let's create our own debugging application. We are going to modify the *Writer* VI from 4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) on page 30 to generate our own logging messages.

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:


100
%lf
Published the value
Log the message to the debugging table with the published value.
Filter level
I32
Published Value
DBL
Publish the data onto the network using RTI DDS
status
Stop Writing
TF

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 4.9.1 Debugging an Application Using the Administration Panel on page 73.
9. Run this new VI and you will see these messages on the administration panel debugging table. The output will be similar to this:


Debugging table

Time	Level	Message
04:14:21 PM Wed 01/13/2016	DL Debug	Published the value 0.512987
04:14:21 PM Wed 01/13/2016	DL Debug	Published the value 0.448052
04:14:21 PM Wed 01/13/2016	DL Debug	Published the value 0.435065
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.435065
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.428571
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.279221
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.272727
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.201299
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.201299
04:14:20 PM Wed 01/13/2016	DL Debug	Published the value 0.162338

4.9.1.2 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.

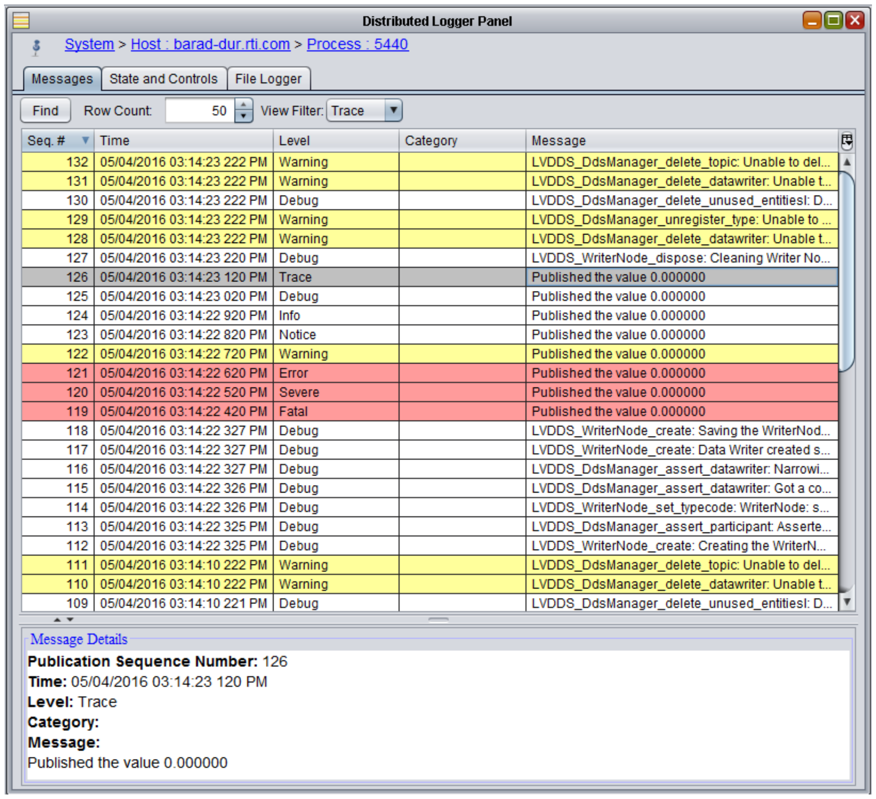
RTI Monitor is included in *RTI Connext DDS Professional*. You can download a free trial from http://www.rti.com/downloads/index.html. For information about *RTI Monitor*, see https://www.rti.-com/products/dds/tools#MONITOR.

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

1. Enable Distributed Logger (see 6.7.1.1 Configuration Section on page 112 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:



Seq. #	Time	Level	Category	Message
132	05/04/2016 03:14:23 222 PM	Warning		LVDDS_DdsManager_delete_topic: Unable to del...
131	05/04/2016 03:14:23 222 PM	Warning		LVDDS_DdsManager_delete_datawriter: Unable t...
130	05/04/2016 03:14:23 222 PM	Debug		LVDDS_DdsManager_delete_unused_entitiesl: D...
129	05/04/2016 03:14:23 222 PM	Warning		LVDDS_DdsManager_unregister_type: Unable to ...
128	05/04/2016 03:14:23 222 PM	Warning		LVDDS_DdsManager_delete_datawriter: Unable t...
127	05/04/2016 03:14:23 220 PM	Debug		LVDDS_WriterNode_dispose: Cleaning Writer No...
126	05/04/2016 03:14:23 120 PM	Trace		Published the value 0.000000
125	05/04/2016 03:14:23 020 PM	Debug		Published the value 0.000000
124	05/04/2016 03:14:22 920 PM	Info		Published the value 0.000000
123	05/04/2016 03:14:22 820 PM	Notice		Published the value 0.000000
122	05/04/2016 03:14:22 720 PM	Warning		Published the value 0.000000
121	05/04/2016 03:14:22 620 PM	Error		Published the value 0.000000
120	05/04/2016 03:14:22 520 PM	Severe		Published the value 0.000000
119	05/04/2016 03:14:22 420 PM	Fatal		Published the value 0.000000
118	05/04/2016 03:14:22 327 PM	Debug		LVDDS_WriterNode_create: Saving the WriterNod...
117	05/04/2016 03:14:22 327 PM	Debug		LVDDS_WriterNode_create: Data Writer created s...
116	05/04/2016 03:14:22 327 PM	Debug		LVDDS_DdsManager_assert_datawriter: Narrowi...
115	05/04/2016 03:14:22 326 PM	Debug		LVDDS_DdsManager_assert_datawriter: Got a co...
114	05/04/2016 03:14:22 326 PM	Debug		LVDDS_WriterNode_set_typecode: WriterNode: s...
113	05/04/2016 03:14:22 325 PM	Debug		LVDDS_DdsManager_assert_participant: Asserte...
112	05/04/2016 03:14:22 325 PM	Debug		LVDDS_WriterNode_create: Creating the WriterN...
111	05/04/2016 03:14:10 222 PM	Warning		LVDDS_DdsManager_delete_topic: Unable to del...
110	05/04/2016 03:14:10 222 PM	Warning		LVDDS_DdsManager_delete_datawriter: Unable t...
109	05/04/2016 03:14:10 221 PM	Debug		LVDDS_DdsManager_delete_unused_entitiesl: D...

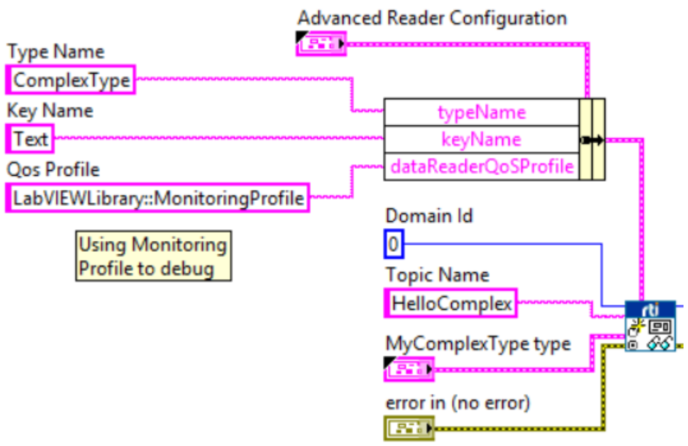
Message Details
Publication Sequence Number: 126
Time: 05/04/2016 03:14:23 120 PM
Level: Trace
Category:
Message:
Published the value 0.000000

4.9.2 Adapting a VI to Use RTI Monitoring Library

Let's begin by opening the *Reader* VI created in 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38: **Tutorial_Read_Cluster.vi**. Or you can use the solution to that lesson mentioned in 4.13 Reviewing Completed Solutions on page 92.

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.



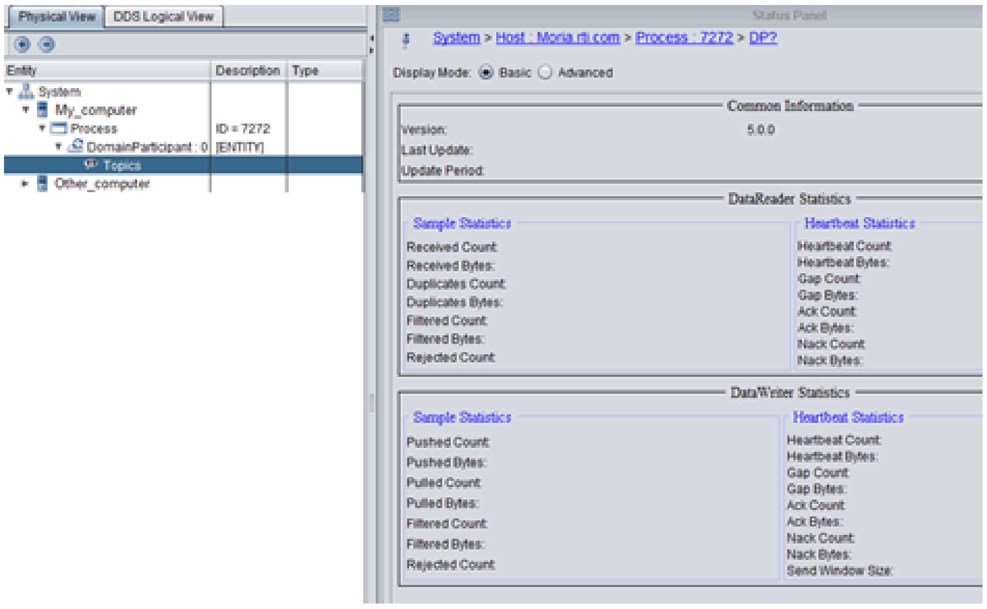
Monitoring can also be enabled by QoS. For more information, refer to Method 2-B: Change the Participant QoS by Specifying the Monitoring Library Create Function Pointer in an Environment Variable in the *Connext Core Libraries User's Manual*.

4.9.2.1 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.

RTI Monitor is included in *RTI Connext DDS Professional*. You can download a free trial from http://www.rti.com/downloads/index.html. For information about *RTI Monitor*, see https://www.rti.com/products/dds/tools#MONITOR.

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.



Entity	Description	Type
System		
My_computer		
Process	ID = 7272	
DomainParticipant : 0	[ENTITY]	
Topics		
Other_computer		

System > Host : Moria.rti.com > Process : 7272 > DP?

Common Information

Version: 5.0.0
Last Update:
Update Period:

DataReader Statistics

Sample Statistics
Received Count:
Received Bytes:
Duplicates Count:
Duplicates Bytes:
Filtered Count:
Filtered Bytes:
Rejected Count:

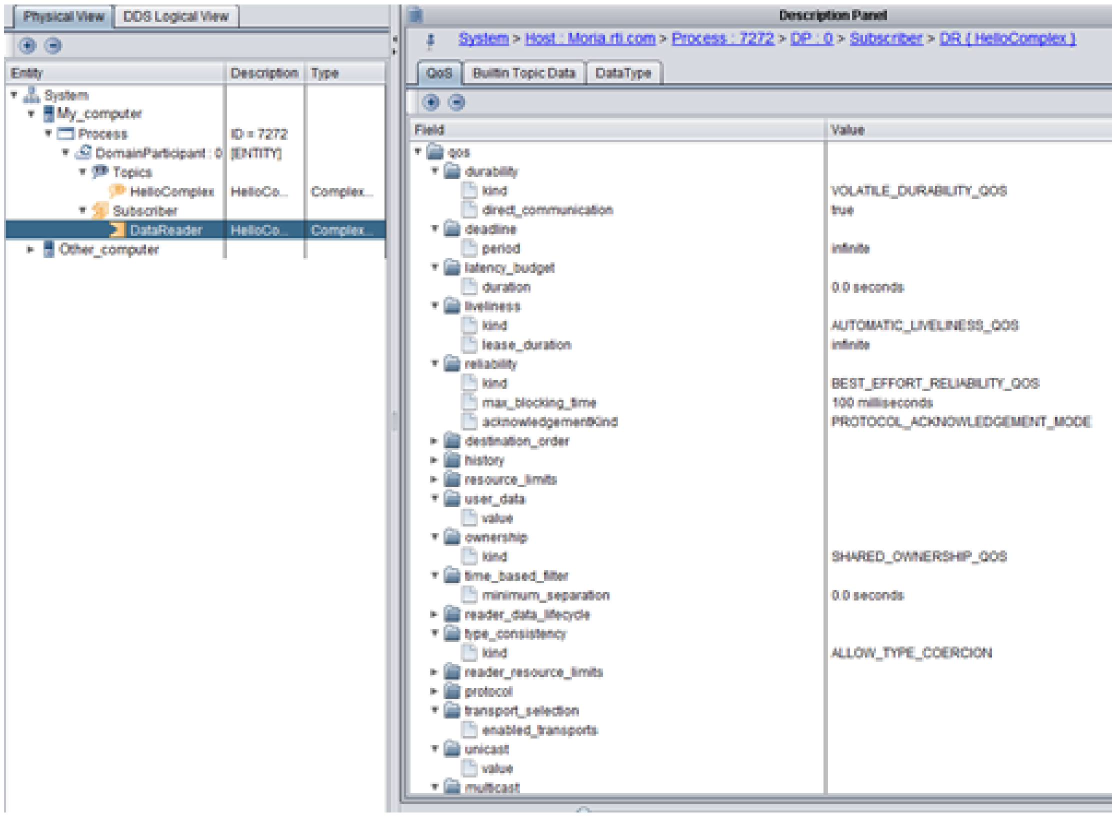
Heartbeat Statistics
Heartbeat Count:
Heartbeat Bytes:
Gap Count:
Gap Bytes:
Ack Count:
Ack Bytes:
Nack Count:
Nack Bytes:

DataWriter Statistics

Sample Statistics
Pushed Count:
Pushed Bytes:
Pulled Count:
Pulled Bytes:
Filtered Count:
Filtered Bytes:
Rejected Count:

Heartbeat Statistics
Heartbeat Count:
Heartbeat Bytes:
Gap Count:
Gap Bytes:
Ack Count:
Ack Bytes:
Nack Count:
Nack Bytes:
Send Window Size:

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.

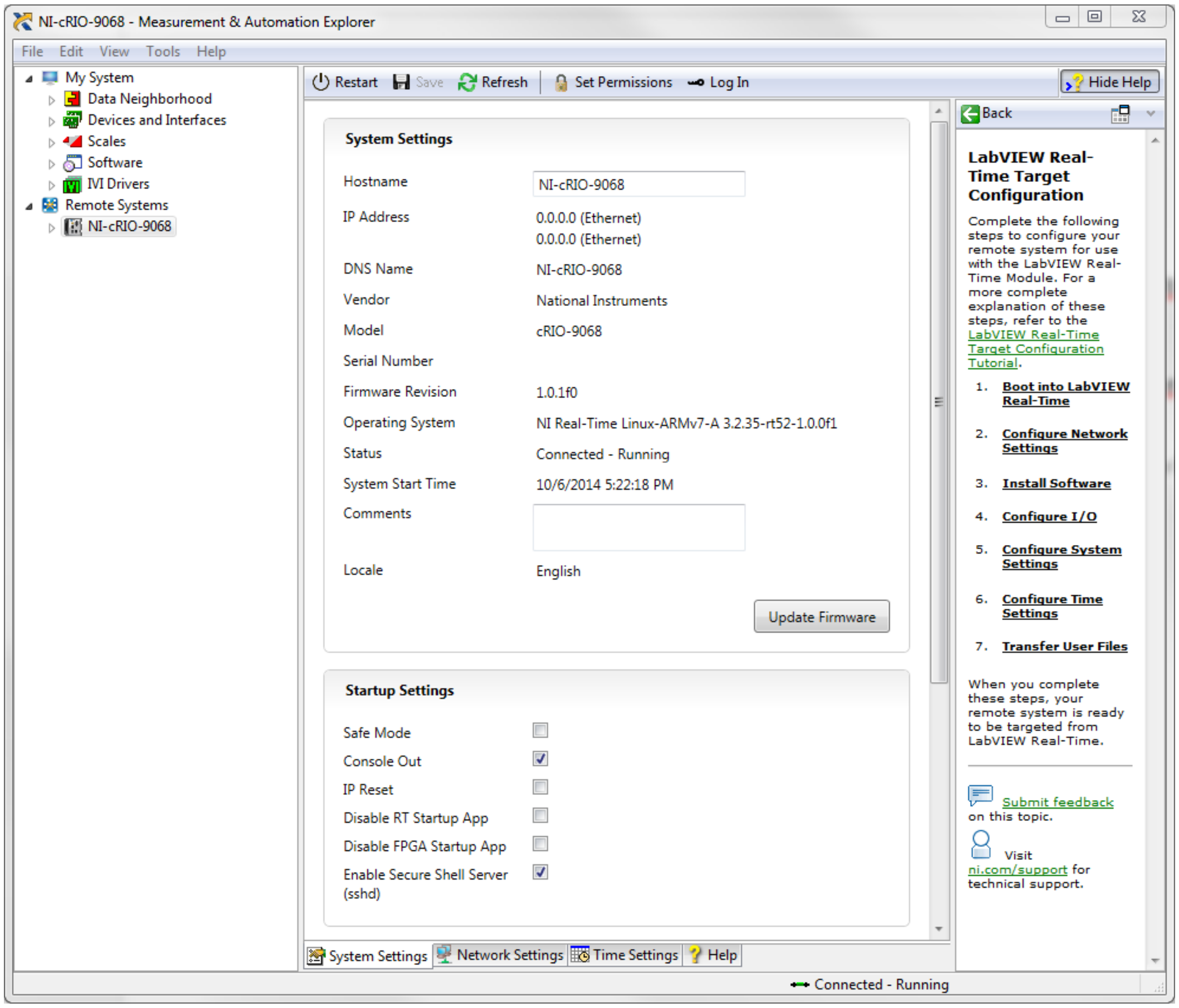

Physical View
DDS Logical View
Description Panel
System > Host : Moria.rti.com > Process : 7272 > DP : 0 > Subscriber > DR (HelloComplex)
QoS
Builtin Topic Data
DataType

Entity	Description	Type
System		
My_computer		
Process	ID = 7272	
DomainParticipant : 0	[ENTITY]	
Topics		
HelloComplex	HelloCo...	Complex...
Subscriber		
DataReader	HelloCo...	Complex...
Other_computer		

Field	Value
qos	
durability	
kind	VOLATILE_DURABILITY_QOS
direct_communication	true
deadline	
period	infinite
latency_budget	
duration	0.0 seconds
liveliness	
kind	AUTOMATIC_LIVELINESS_QOS
lease_duration	infinite
reliability	
kind	BEST_EFFORT_RELIABILITY_QOS
max_blocking_time	100 milliseconds
acknowledgementKind	PROTOCOL_ACKNOWLEDGEMENT_MODE
destination_order	
history	
resource_limits	
user_data	
value	
ownership	
kind	SHARED_OWNERSHIP_QOS
time_based_filter	
minimum_separation	0.0 seconds
reader_data_lifecycle	
type_consistency	
kind	ALLOW_TYPE_COERCION
reader_resource_limits	
protocol	
transport_selection	
enabled_transports	
unicast	
value	
multicast	

4.10 Lesson 10–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.


NI-cRIO-9068 - Measurement & Automation Explorer
File Edit View Tools Help
My System
Data Neighborhood
Devices and Interfaces
Scales
Software
IVI Drivers
Remote Systems
NI-cRIO-9068
Restart Save Refresh Set Permissions Log In
Hide Help
Back
Update Firmware
System Settings Network Settings Time Settings Help
Connected - Running

System Settings

Hostname	NI-cRIO-9068
IP Address	0.0.0.0 (Ethernet) 0.0.0.0 (Ethernet)
DNS Name	NI-cRIO-9068
Vendor	National Instruments
Model	cRIO-9068
Serial Number	
Firmware Revision	1.0.1f0
Operating System	NI Real-Time Linux-ARMv7-A 3.2.35-rt52-1.0.0f1
Status	Connected - Running
System Start Time	10/6/2014 5:22:18 PM
Comments	
Locale	English

Startup Settings

Safe Mode	☐
Console Out	✓
IP Reset	☐
Disable RT Startup App	☐
Disable FPGA Startup App	☐
Enable Secure Shell Server (sshd)	✓

LabVIEW Real-Time Target Configuration

Complete the following steps to configure your remote system for use with the LabVIEW Real-Time Module. For a more complete explanation of these steps, refer to the LabVIEW Real-Time Target Configuration Tutorial.

1. Boot into LabVIEW Real-Time
2. Configure Network Settings
3. Install Software
4. Configure I/O
5. Configure System Settings
6. Configure Time Settings
7. Transfer User Files

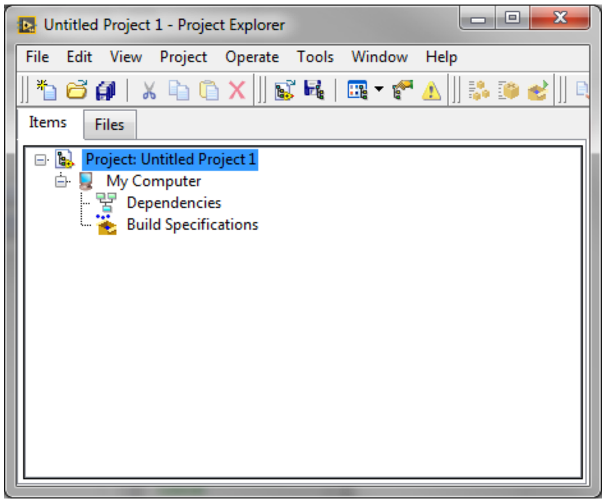
When you complete these steps, your remote system is ready to be targeted from LabVIEW Real-Time.

Submit feedback on this topic.

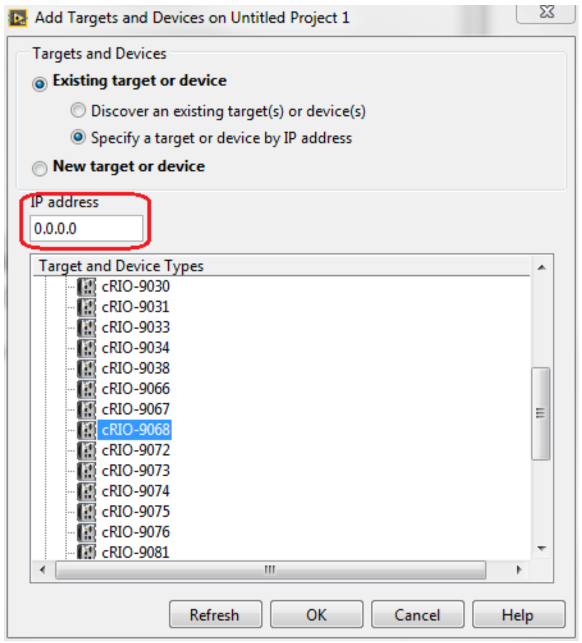
Visit ni.com/support for technical support.

2. Follow the installation instructions in 1.2.1 Installing RTI DDS Toolkit Support Files on a Target on page 4.

3. Create an empty project in LabVIEW by choosing **File, New Project** or **File, Create Project**, depending on your LabVIEW version.


Untitled Project 1 - Project Explorer
File Edit View Project Operate Tools Window Help
Items
Files
Project: Untitled Project 1
My Computer
Dependencies
Build Specifications

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.


Add Targets and Devices on Untitled Project 1

Targets and Devices

[✓ **Existing target or device**]

Discover an existing target(s) or device(s)

[✓ Specify a target or device by IP address]

New target or device

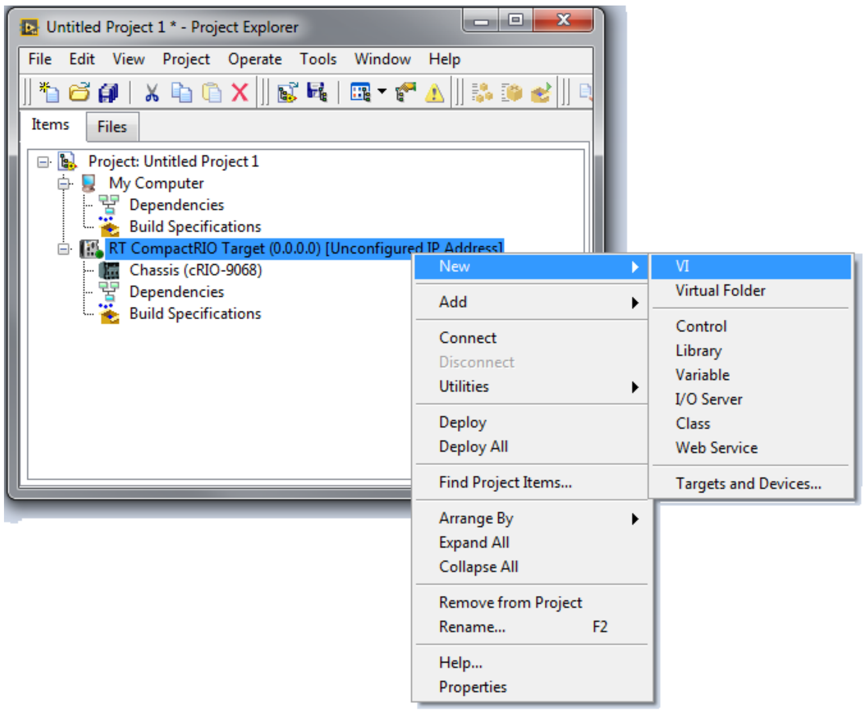
IP address: 0.0.0.0

Target and Device Types
- cRIO-9030
- cRIO-9031
- cRIO-9033
- cRIO-9034
- cRIO-9038
- cRIO-9066
- cRIO-9067
- cRIO-9068
- cRIO-9072
- cRIO-9073
- cRIO-9074
- cRIO-9075
- cRIO-9076
- cRIO-9081

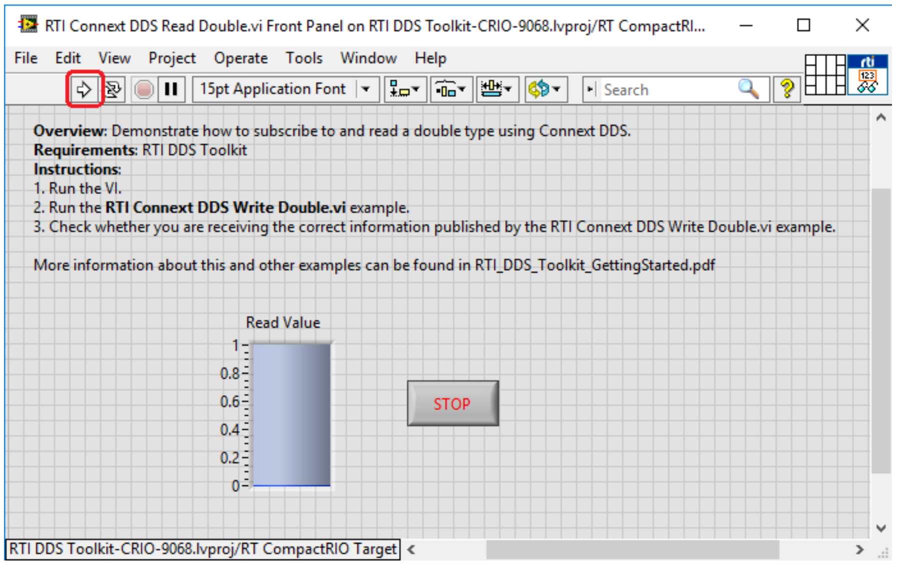
5. Select **Existing target or device** and **Specify a target or device by IP address**. Set the correct IP address. Select your device from the list. You can find a list of supported platforms in the 'Supported Platforms' section of the *Release Notes*. Click **OK**.

   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…**.



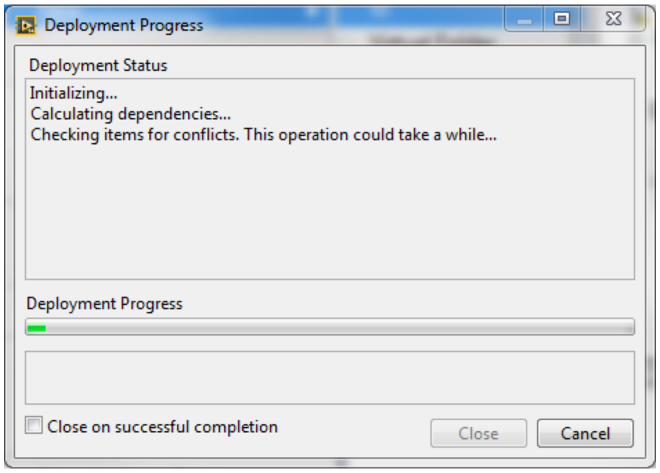
7. Create your application using *RTI DDS Toolkit* as mentioned in the previous lessons. Save it and the project.
8. Once you are finished, run your VI as usual by clicking on the white arrow.


RTI Connext DDS Read Double.vi Front Panel on RTI DDS Toolkit-CRIO-9068.lvproj/RT CompactRI...
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Read Value
1
0.8
0.6
0.4
0.2
0
STOP
RTI DDS Toolkit-CRIO-9068.lvproj/RT CompactRIO Target

Overview: Demonstrate how to subscribe to and read a double type using Connext DDS.
Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run the **RTI Connext DDS Write Double.vi** example.
3. Check whether you are receiving the correct information published by the RTI Connext DDS Write Double.vi example.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

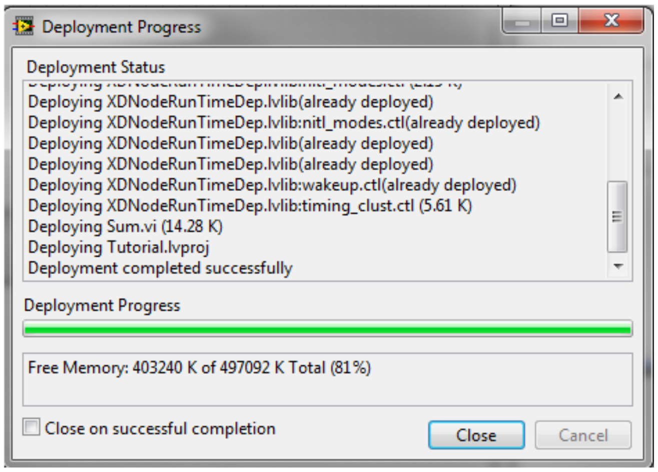
9. LabVIEW will show the Deployment Progress window and will send the VI to your target. This process may take a while, depending on your VI's complexity.



```
Initializing...
Calculating dependencies...
Checking items for conflicts. This operation could take a while...
```

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:



```
Deploying XDNodeRunTimeDep.lvlib(already deployed)
Deploying XDNodeRunTimeDep.lvlib:nitl_modes.ctl(already deployed)
Deploying XDNodeRunTimeDep.lvlib(already deployed)
Deploying XDNodeRunTimeDep.lvlib(already deployed)
Deploying XDNodeRunTimeDep.lvlib:wakeup.ctl(already deployed)
Deploying XDNodeRunTimeDep.lvlib:timing_clust.ctl (5.61 K)
Deploying Sum.vi (14.28 K)
Deploying Tutorial.lvproj
Deployment completed successfully
```

```
Free Memory: 403240 K of 497092 K Total (81%)
```

11. Click **Close** and work with your VI as you normally would.

4.11 Lesson 11–Using Security with RTI DDS Toolkit (Windows only)

This example is based on the *RTI Shapes Demo* example in 4.6 Lesson 6—Using Keyed Types (RTI Shapes Demo) on page 55.

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.

Security features are only supported in Windows platforms. Trying to use security in an NI Linux target will result in error 5113.

4.11.1 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 6.0.0 or later can be used to graphically show this communication. You can ask for a *Shapes Demo* trial.

We will use these topics:

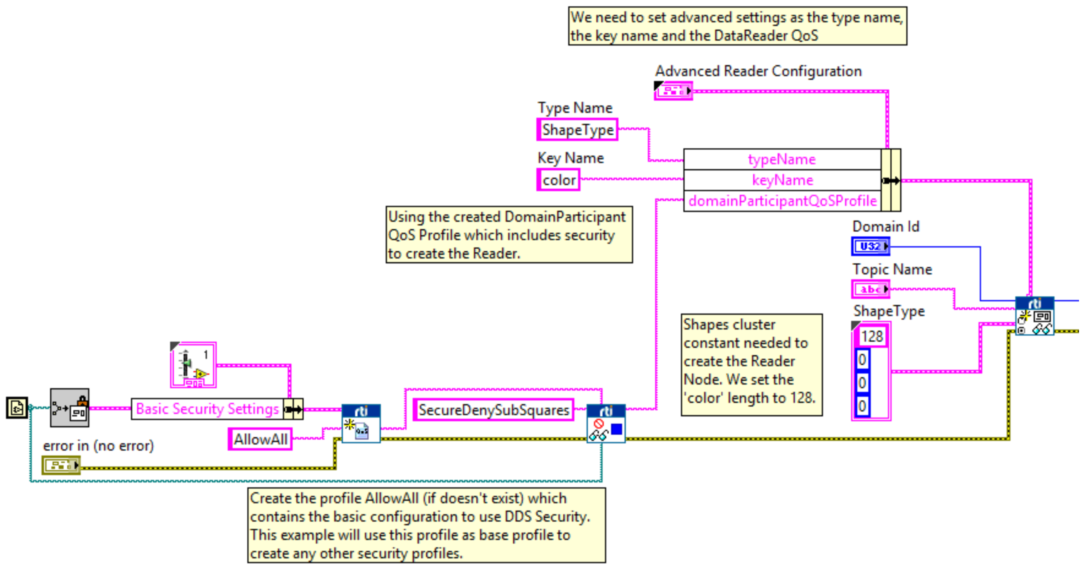
- Square
- Circle
- Triangle

And these security profiles:

- AllowAll
- SecureDenySubSquares

These profiles can be created using the instructions in 6.8.1 Managing Custom Security Profiles with the Security Panel (Windows only) on page 121 as well as the provided subVIs described in 6.8.2 Managing Custom Security Profiles with SubVIs on page 125.

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.

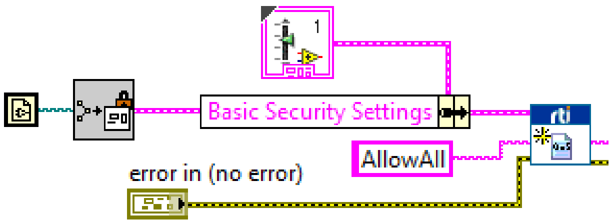

We need to set advanced settings as the type name, the key name and the DataReader QoS
Advanced Reader Configuration
Type Name
ShapeType
Key Name
color
typeName
keyName
domainParticipantQoSProfile
Using the created DomainParticipant QoS Profile which includes security to create the Reader.
Domain Id
Topic Name
ShapeType
Shapes cluster constant needed to create the Reader Node. We set the 'color' length to 128.
128
0
0
0
Basic Security Settings
AllowAll
SecureDenySubSquares
error in (no error)
Create the profile AllowAll (if doesn't exist) which contains the basic configuration to use DDS Security. This example will use this profile as base profile to create any other security profiles.

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 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 **signed_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:

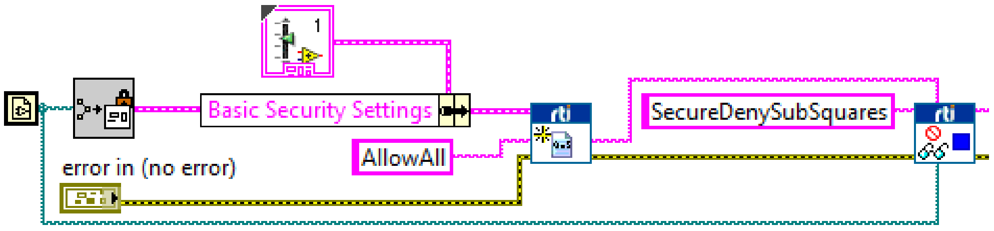
```
    <default>ALLOW</default>
```



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 private key **ecdsa01Peer03Key.pem** used in the example is encrypted. The password is set in **Create SecureDenySubSquares.vi**. For details on setting passwords for encrypted private keys, see 6.8.2 Managing Custom Security Profiles with SubVIs on page 125. 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**.

4.11.2 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 subfolder 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 **ecdsa01Peer03 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**.

4.11.3 Main Scenarios

There are three 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:

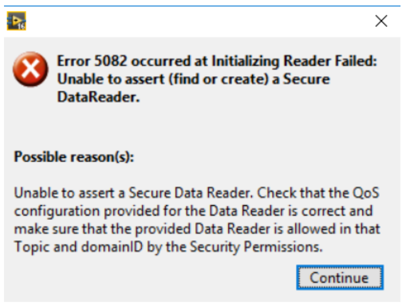
- *Writer*: **AllowAll**
- *Reader*: **SecureDenySubSquares**

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 or Circle, the communication will be fine because no restrictions apply to this topic.

For more information about creating your own governance rules, refer to Hands-On 2: Defining Your System's Security Requirements in the *RTI Security Plugins Getting Started Guide*. The shipped p7s files provided in the **SecurityShapesDemo\cert** folder are signed and will not work if you try to modify them.

4.11.4 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.



Error 5082 occurred at Initializing Reader Failed: Unable to assert (find or create) a Secure DataReader.

Possible reason(s):

Unable to assert a Secure Data Reader. Check that the QoS configuration provided for the Data Reader is correct and make sure that the provided Data Reader is allowed in that Topic and domainID by the Security Permissions.

- 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.

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

4.12 Lesson 12–Reading Multiple Samples at a Time

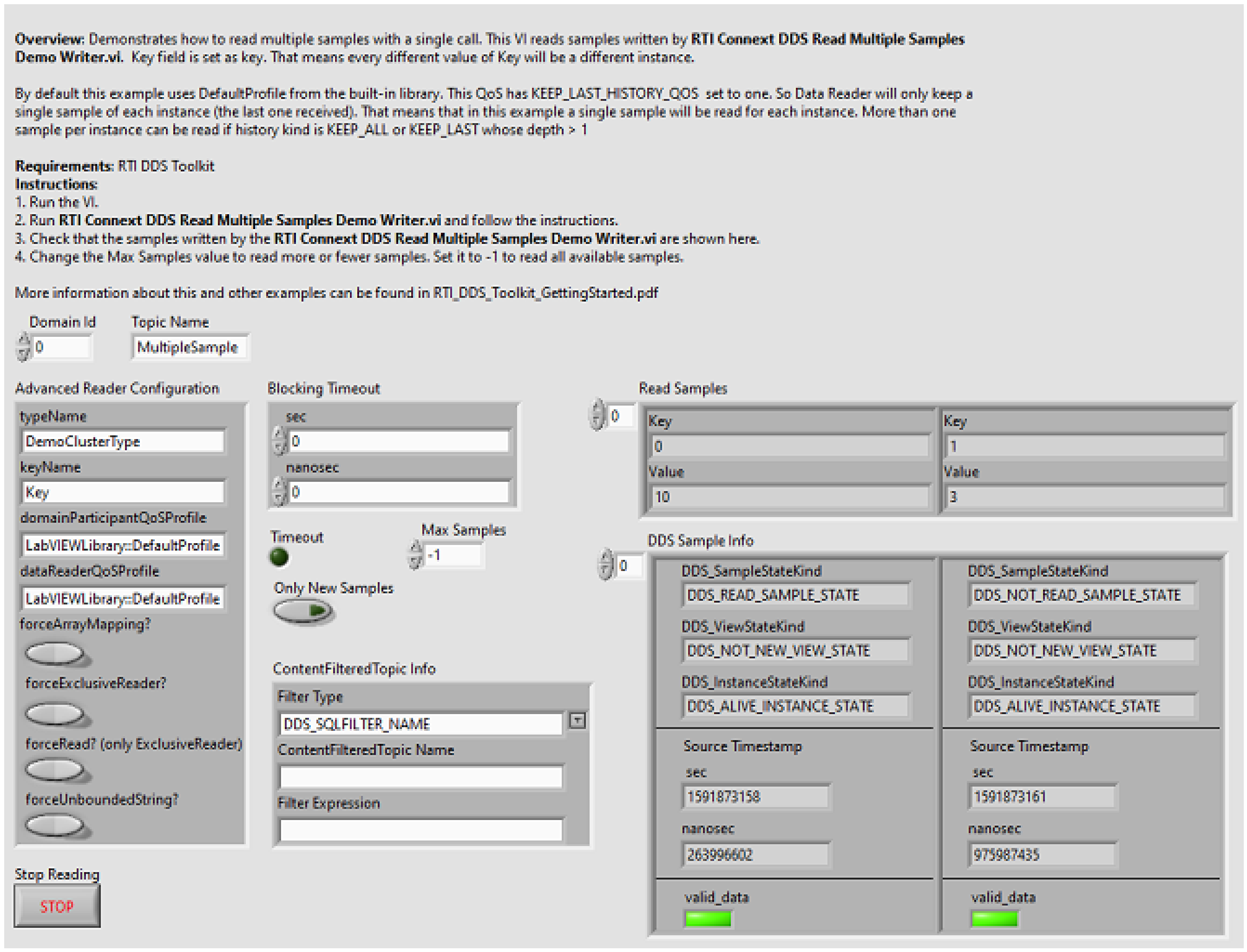
This lesson shows how to read multiple samples in a single call.

In addition to **<Type Name> Read.vi** and **<Type Name> Write.vi**, the *ComplexType Generator* also generates a VI called **<Type Name> Read Multiple Samples.vi**. This VI is very similar to **<Type Name> Read.vi**. The difference is that **<Type Name> Multiple Samples.vi** reads multiple samples in a single call and returns them in a LabVIEW Array.

Open the **ReadMultipleSamplesDemo**. This example has two main files:

- **RTI Connext DDS Read Multiple Samples Demo Reader.vi**
- **RTI Connext DDS Read Multiple Samples Demo Writer.vi**

RTI Connext DDS Read Multiple Samples Demo Reader.vi:



Overview: Demonstrates how to read multiple samples with a single call. This VI reads samples written by **RTI Connext DDS Read Multiple Samples Demo Writer.vi.** Key field is set as key. That means every different value of Key will be a different instance.

By default this example uses DefaultProfile from the built-in library. This QoS has KEEP_LAST_HISTORY_QOS set to one. So Data Reader will only keep a single sample of each instance (the last one received). That means that in this example a single sample will be read for each instance. More than one sample per instance can be read if history kind is KEEP_ALL or KEEP_LAST whose depth > 1

Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Run **RTI Connext DDS Read Multiple Samples Demo Writer.vi** and follow the instructions.
3. Check that the samples written by the **RTI Connext DDS Read Multiple Samples Demo Writer.vi** are shown here.
4. Change the Max Samples value to read more or fewer samples. Set it to -1 to read all available samples.

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Domain Id: 0
Topic Name: MultipleSample

Advanced Reader Configuration
typeName: DemoClusterType
keyName: Key
domainParticipantQoSProfile: LabVIEWLibrary::DefaultProfile
dataReaderQoSProfile: LabVIEWLibrary::DefaultProfile
forceArrayMapping?
forceExclusiveReader?
forceRead? (only ExclusiveReader)
forceUnboundedString?

Blocking Timeout
sec: 0
nanosec: 0

Timeout
Max Samples: -1
Only New Samples

ContentFilteredTopic Info
Filter Type: DDS_SQLFILTER_NAME
ContentFilteredTopic Name:
Filter Expression:

Stop Reading
STOP

Read Samples: 0

Key	0	1
Value	10	3

DDS Sample Info: 0

DDS_SampleStateKind	DDS_READ_SAMPLE_STATE	DDS_NOT_READ_SAMPLE_STATE
DDS_ViewStateKind	DDS_NOT_NEW_VIEW_STATE	DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind	DDS_ALIVE_INSTANCE_STATE	DDS_ALIVE_INSTANCE_STATE
Source Timestamp sec	1591873158	1591873161
Source Timestamp nanosec	263996602	975987435
valid_data	🟩	🟩

RTI Connext DDS Read Multiple Samples Demo Reader.vi has three relevant items:

- **Read Samples** shows the different instances received.
- **DDS Sample Info** shows the received Sample Info for each sample. The output is an array of DDS Sample Infos with the same size as the number of samples returned by the call.
- **Max Samples** is the maximum number of samples to be read. To read all available samples set Max Samples to -1.

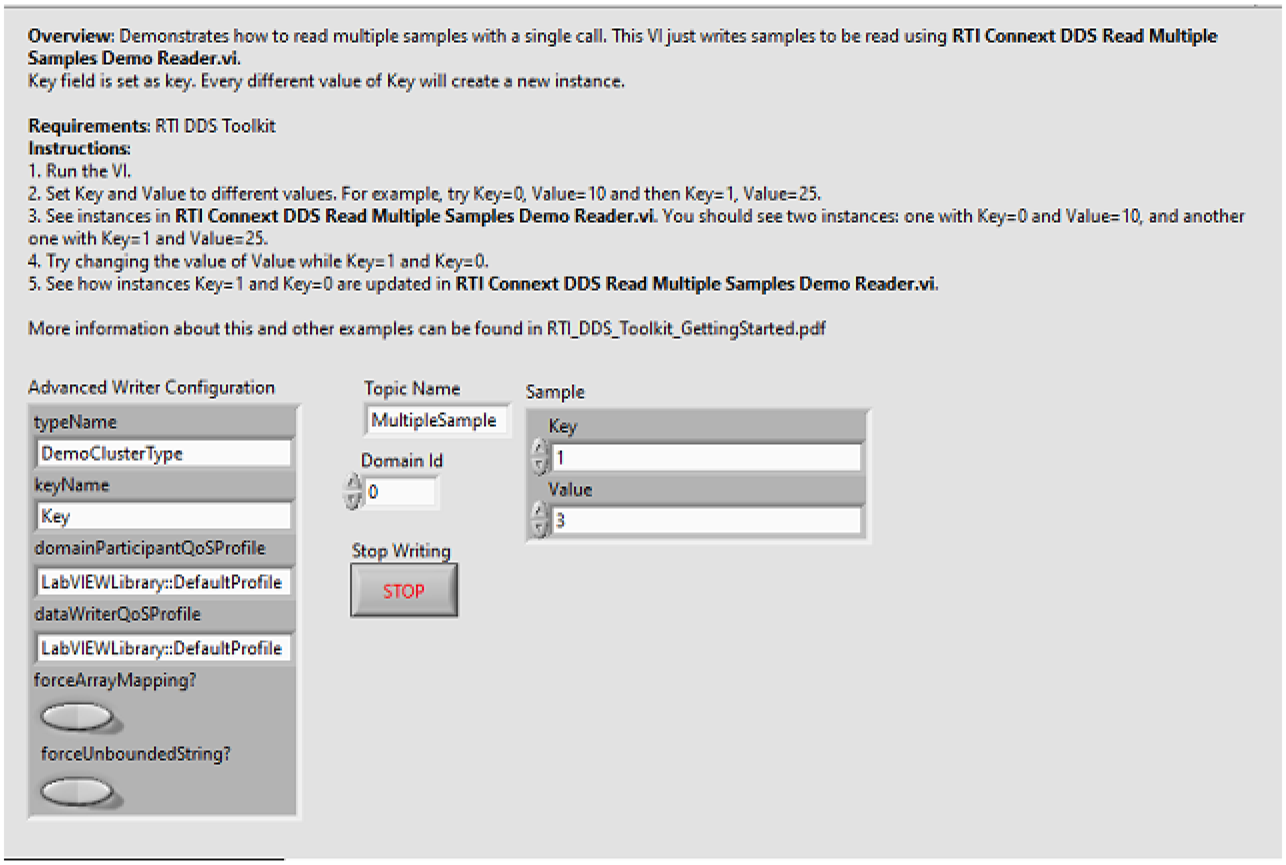
The type used in the example is keyed. The key field is called **Key**. By default, the History QoS kind is **KEEP_LAST** and the **depth** is 1. This means Readers and Writers will only store in memory one sample of each instance, therefore only one sample can be read at a time. If several instances are sent, the DDS Reader will keep one sample of each instance in its queue.

Follow these steps:

1. Run the **RTI Connext DDS Read Multiple Samples Demo Reader.vi**.
2. Run the **RTI Connext DDS Read Multiple Samples Demo Writer.vi** and follow the instructions in the VI (which are also described below).
3. Check that the samples written by **RTI Connext DDS Read Multiple Samples Demo Writer.vi** appear in the Reader.
4. Change **Max Samples** to read more or fewer samples. Set it to -1 to read all available samples.

RTI Connext DDS Read Multiple Samples Demo Writer.vi writes arrays of samples every two seconds. The **Sample** array control is the array of samples to be sent. The type used is keyed. The key field is **ID**.

RTI Connext DDS Read Multiple Samples Demo Writer.vi:



Overview: Demonstrates how to read multiple samples with a single call. This VI just writes samples to be read using **RTI Connext DDS Read Multiple Samples Demo Reader.vi.**
Key field is set as key. Every different value of Key will create a new instance.

Requirements: RTI DDS Toolkit
Instructions:
1. Run the VI.
2. Set Key and Value to different values. For example, try Key=0, Value=10 and then Key=1, Value=25.
3. See instances in **RTI Connext DDS Read Multiple Samples Demo Reader.vi.** You should see two instances: one with Key=0 and Value=10, and another one with Key=1 and Value=25.
4. Try changing the value of Value while Key=1 and Key=0.
5. See how instances Key=1 and Key=0 are updated in **RTI Connext DDS Read Multiple Samples Demo Reader.vi.**

More information about this and other examples can be found in RTI_DDS_Toolkit_GettingStarted.pdf

Advanced Writer Configuration
typeName: DemoClusterType
keyName: Key
domainParticipantQoSProfile: LabVIEWLibrary::DefaultProfile
dataWriterQoSProfile: LabVIEWLibrary::DefaultProfile
forceArrayMapping?
forceUnboundedString?

Topic Name: MultipleSample
Domain Id: 0

Sample
Key: 1
Value: 3

Follow these steps for the Writer:

1. Run the VI.
2. Set Key and Value to different values. For example, try Key=0, Value=10, then Key=1, Value-e=25.
3. See the instances in **RTI Connext DDS Read Multiple Samples Demo Reader.vi**. You should see two instances: one with Key=0 and Value=10, and another one with Key=1 and Value=25.
4. Try changing the value of Value while Key=1 and Key=0.
5. See how instances Key=1 and Key=0 are updated in **RTI Connext DDS Read Multiple Samples Demo Reader.vi**.

4.13 Reviewing Completed Solutions

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

- 4.1 Lesson 1—Using DDS to Publish and Subscribe to Simple Data (Numeric) on page 30

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\NumberDemo
- 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ClusterDemo
- 4.3 Lesson 3—Blocking Reads on page 45

  \LabVIEW 20xx\LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\BlockingReadDemo
- 4.4 Lesson 4—Filtering Data on page 47

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ClusterDemo
- 4.5 Lesson 5—Reading Only New Samples on page 52

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ClusterDemo
- 4.6 Lesson 6—Using Keyed Types (RTI Shapes Demo) on page 55

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ShapesDemo
- 4.8 Lesson 8—Reading All Samples (Reliable Communication) on page 64

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\ReadAllDemo
- 4.9 Lesson 9—Debugging Your RTI Connext DDS Application on page 73

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\LogMessagesDemo

  \LabVIEW 20xx\examples\RTI DDS Toolkit\Examples\MonitoringDemo
- 4.10 Lesson 10—Using RTI DDS Toolkit on NI Targets (cRIO-9068 Example) on page 79

  \LabVIEW 20xx\examples\RTI DDS Toolkit\cRIO-9068Project
  (Note: This project is compatible with LabVIEW 2013 and higher)
- 4.11 Lesson 11—Using Security with RTI DDS Toolkit (Windows only) on page 83

  \LabVIEW 20xx\examples\RTI DDS Toolkit\SecurityShapesDemo
- 4.12 Lesson 12—Reading Multiple Samples at a Time on page 88

  \LabVIEW 20xx\examples\RTI DDS Toolkit\ReadAndWriteArrayDemo

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**[1]**/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 builtin profiles, use their library name and profile name (for instance, BuiltinQosLib::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,

[1]On 64-bit systems, the folder is "Program Files (x86)"

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].

  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[1]/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[2]/National Instruments/Shared/LabVIEW Run-Time/2012/**.

[1]On 64-bit systems, the folder is “Program Files (x86)”

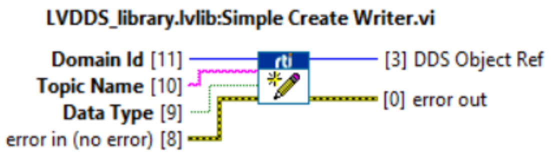
[2]On 64-bit systems, the folder is “Program Files (x86)”

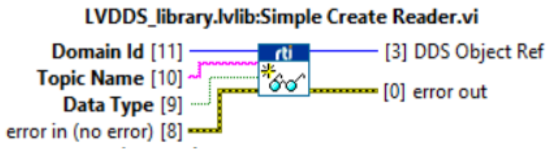
Chapter 6 Advanced Concepts and Settings

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

When configuring an *RTI Connext* 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 on page 94). 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.[1] 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.

[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 6.3.1 Using the RTI DDS ComplexType Generator on page 99 for details.

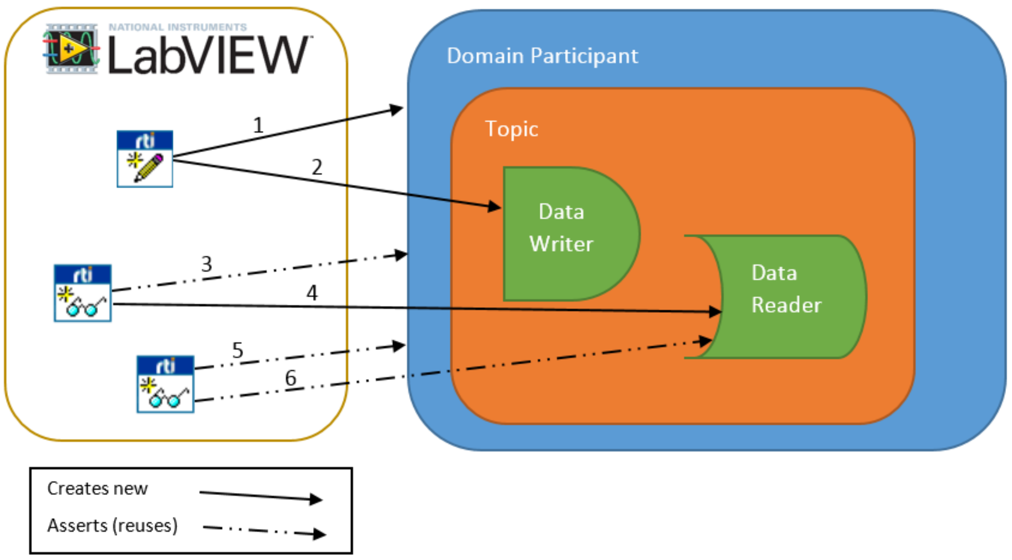
6.1 Default Configuration: DDS Entities Created by ‘Simple Create’ SubVIs

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**.

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. DDS entities will be removed from memory when the time to delete inactive DDS entities expires (refer to **Timeout to delete inactive DDS entities** in the Configuration Section), or when the last VI that uses *DDS Toolkit* subVIs in execution stops.

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.

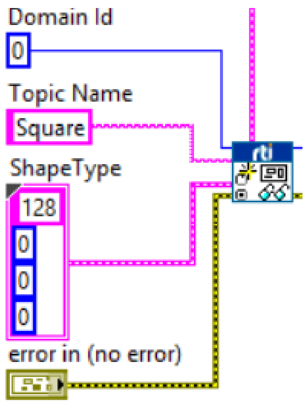
To fix this, we can provide the maximum length of the array by typing a number as the value of the string when we create the DataReader/DataWriter. So a string whose text is “75” will be creates as a DDS String of 75 characters. This is the representation of the equivalent IDL:

```
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



```
struct ShapeType {
    string<128> color; //@key
    long x;
    long y;
    long shapesize;
};
```

6.3 Working with Custom Types

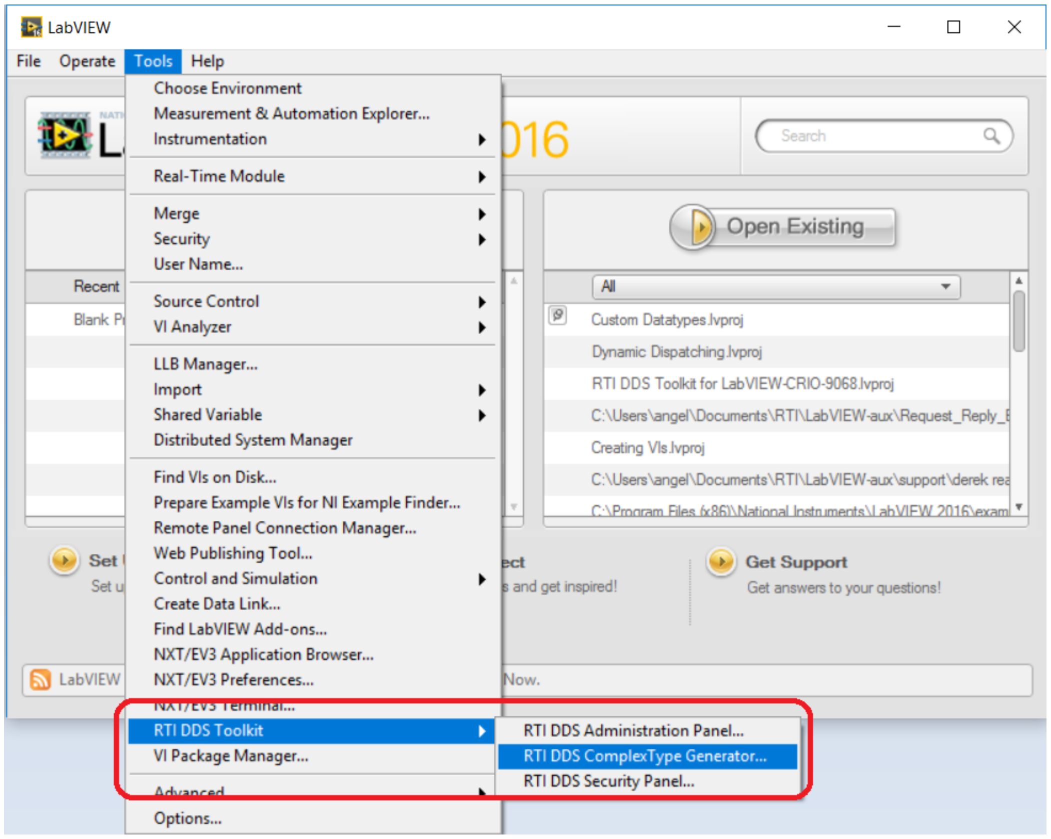
This section describes two ways to generate custom type VIs:

- 6.3.1 Using the RTI DDS ComplexType Generator below
- 6.3.2 Using the VI called 'DDS Generate Custom Type VIs' on page 102

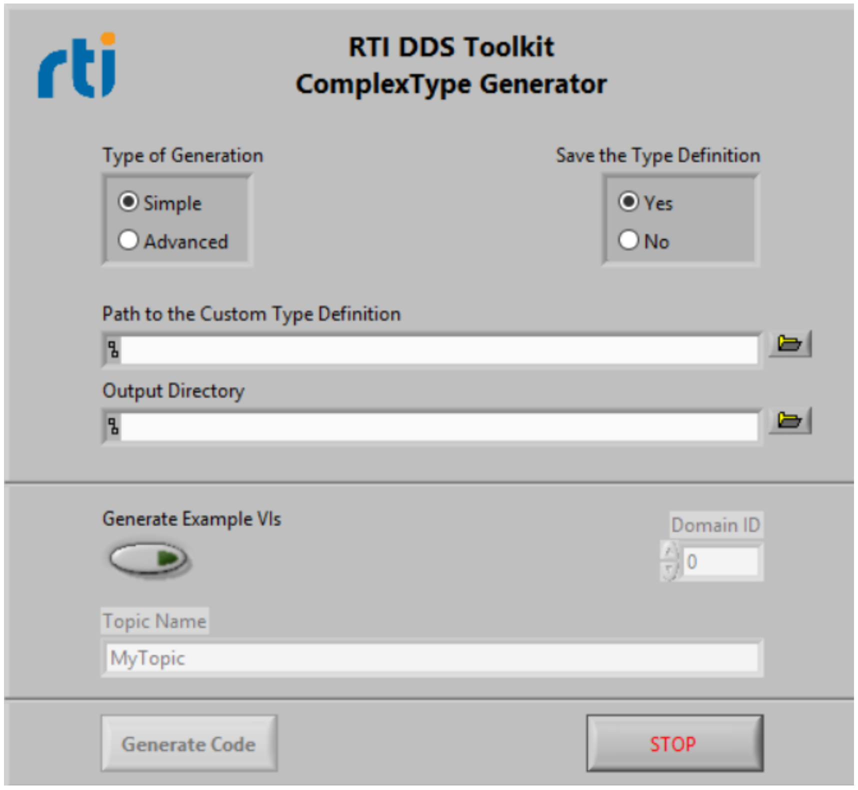
6.3.1 Using the RTI DDS ComplexType Generator

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**).


LabVIEW
File
Operate
Tools
Help
Choose Environment
Measurement & Automation Explorer...
Instrumentation
Real-Time Module
Merge
Security
User Name...
Source Control
VI Analyzer
LLB Manager...
Import
Shared Variable
Distributed System Manager
Find VIs on Disk...
Prepare Example VIs for NI Example Finder...
Remote Panel Connection Manager...
Web Publishing Tool...
Control and Simulation
Create Data Link...
Find LabVIEW Add-ons...
NXT/EV3 Application Browser...
NXT/EV3 Preferences...
NXT/EV3 Terminal...
RTI DDS Toolkit
VI Package Manager...
Advanced
Options...
RTI DDS Administration Panel...
RTI DDS ComplexType Generator...
RTI DDS Security Panel...
Search
Open Existing
All
Custom Datatypes.lvproj
Dynamic Dispatching.lvproj
RTI DDS Toolkit for LabVIEW-CRIO-9068.lvproj
C:\Users\angel\Documents\RTI\LabVIEW-aux\Request_Reply_E
Creating VIs.lvproj
C:\Users\angel\Documents\RTI\LabVIEW-aux\support\derek rea
C:\Program Files (x86)\National Instruments\LabVIEW 2016\exam
Get Support
Get answers to your questions!

Let's take a look at the ComplexType Generator Wizard:



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 6.4 Configuring Advanced Writer Settings on page 103 and 6.5 Configuring Advanced Reader Settings on page 104 to learn more about the advanced creation subVIs).
- **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*
- *ComplexType Read Array*
- *ComplexType Read Multiple Samples*

If **Generation of Example VIs** is enabled, the following subVIs will also be generated:

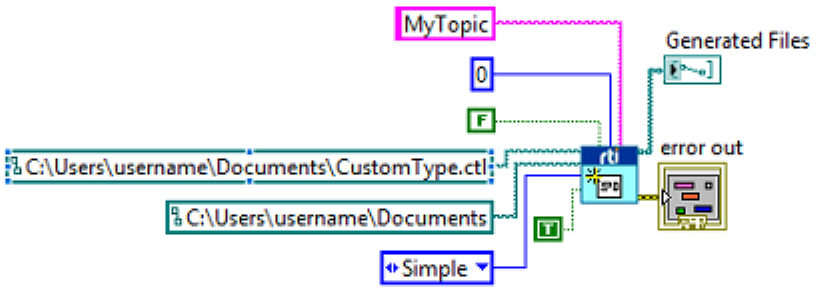
- *ComplexType Reader Example*
- *ComplexType Writer Example*
- *ComplexType Array Reader Example*
- *ComplexType Array Writers Example*
- *Complex Type Read Multiple Samples Example*

The *ComplexType Generator* will check for unsupported types and report an error if it finds any of these issues:

- Enums with a representation that is not U32
- Rings with a representation that is not I32
- Unlabeled members
- Duplicate member names in the same cluster

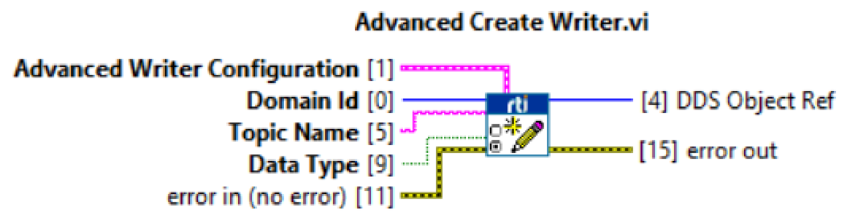
6.3.2 Using the VI called 'DDS Generate Custom Type VIs'

There is also a VI called *DDS Generate Custom Type VIs* can be used the same way as the ComplexType Generator to create custom types VIs. The parameters are the same as those required by the ComplexType Generator.

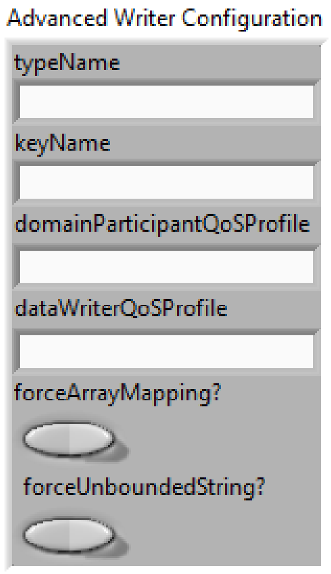

MyTopic
0
F
C:\Users\username\Documents\CustomType.ctl
C:\Users\username\Documents
Simple
T
rti
Generated Files
error out

6.4 Configuring Advanced Writer Settings

In the Writer subpalette, you can find an *Advanced Create Writer*.[1] 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**.



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 Supported Data Types and Corresponding IDL on page 147).
- **keyName**: List of fields of a data type that will be marked as key (see 4.6 Lesson 6—Using Keyed Types (RTI Shapes Demo) on page 55 and 4.7 Lesson 7—Used Nested and Multiple Keys on page 61).
- **domainParticipantQoSProfile**: Custom Security Profile or 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.[2]
- **dataWriterQoSProfile**: Custom Security Profile or fully qualified name (Library::Profile) that will be used as a QoS profile when creating the DataWriter.
- **forceArrayMapping?**: By default, one-dimensional LabVIEW arrays are mapped as DDS sequences. If you need your data to use DDS arrays, set this flag to true. This will affect all

[1]For complex types, use the ComplexTypes Generator in the **Tools/RTI DDS Toolkit** menu. See 6.3.1 Using the RTI DDS ComplexType Generator on page 99.

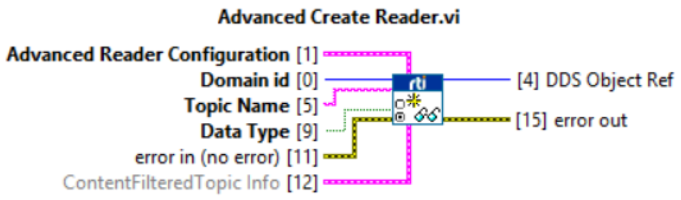
[2]Read this article on the creation of multiple DomainParticipants: https://community.rti.com/best-practices/create-few-domain-participants-possible

LabVIEW arrays in the data. (Note: Multi-dimensional arrays are mapped as DDS arrays, see 6.10 Setting Up Arrays on page 129.)

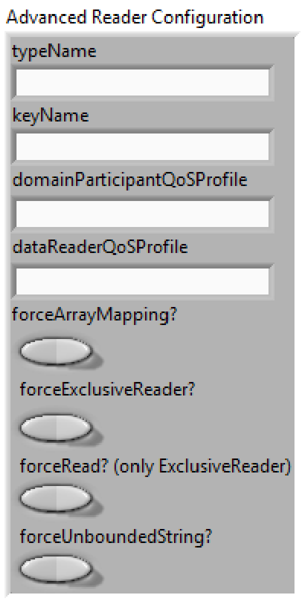
- **forceUnboundedString?**: By default, strings are created with a length of 1024 characters. If this flag is set to true, all strings are created as unbounded (their maximum length corresponds to the maximum 32-bit integer). This configuration optimizes the sample size, sending only the actual data while removing the 1024-character limitation in previous versions of *RTI DDS Toolkit*. This will affect all strings in the data.

6.5 Configuring Advanced Reader Settings

In the Reader subpalette, you can find an *Advanced Create Reader* subVI.[1] 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 Supported Data Types and Corresponding IDL on page 147).
- **keyName**: List of fields in a data type that will be marked as key (see 4.6 Lesson 6—Using Keyed Types (RTI Shapes Demo) on page 55 and 4.7 Lesson 7—Used Nested and Multiple Keys on page 61).
- **domainParticipantQoSProfile**: Custom Security Profile or 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

[1]For complex types, use the ComplexTypes Generator in the Tools/RTI DDS Toolkit menu. See 6.3.1 Using the RTI DDS ComplexType Generator on page 99 for details.

the provided QoS profile. Creating many different DDS DomainParticipants may affect performance.

- **dataReaderQoSProfile**: Custom Security Profile or fully qualified name (Library::Profile) that will be used as the QoS profile when creating the DataReader.
- **forceArrayMapping?**: By default, LabVIEW arrays are mapped as DDS sequences. If you need your data to use DDS arrays, set this flag to true. This will affect all LabVIEW arrays in the data.
- **forceExclusiveReader?**: By default, *Reader* nodes of the same topic (and with the same QoS profile) share a DataReader. To avoid this behavior, set this flag to true and a new DataReader will be created. If you need all your *Reader* nodes to have their own DataReader, make sure all of them are created setting this flag to true.
- **forceRead?**: By default, exclusive *Readers* call to the function **take** when getting the data. This allows you to use the Strict Reliable QoS profile. If you want to use **read** instead, set this flag to true.
- **forceUnboundedString?**: By default, strings are created with a length of 1,024 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, receiving only the actual data while removing the 1,024-character limitation in previous versions of *RTI DDS Toolkit*. This will affect all strings in the data.

If you need to use Strict Reliability QoS profile, make sure your *Reader* node is exclusive and **forceRead** is set to false (the default value).

6.6 Working with Instance State Kind

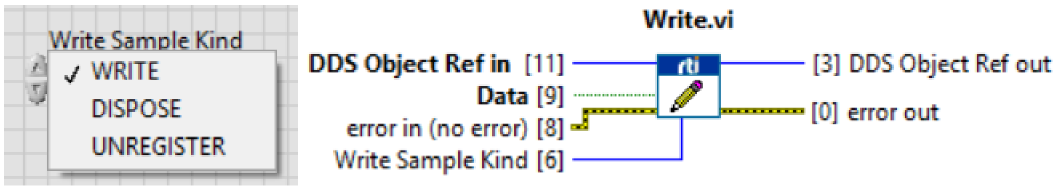
The “DDS Sample Info” structure contains the value of the DDS_InstanceStateKind (aka Instance State). The Instance State is a per-instance concept. An instance is a unique element of a specific DataType within a Topic, described by unique values of key fields.

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

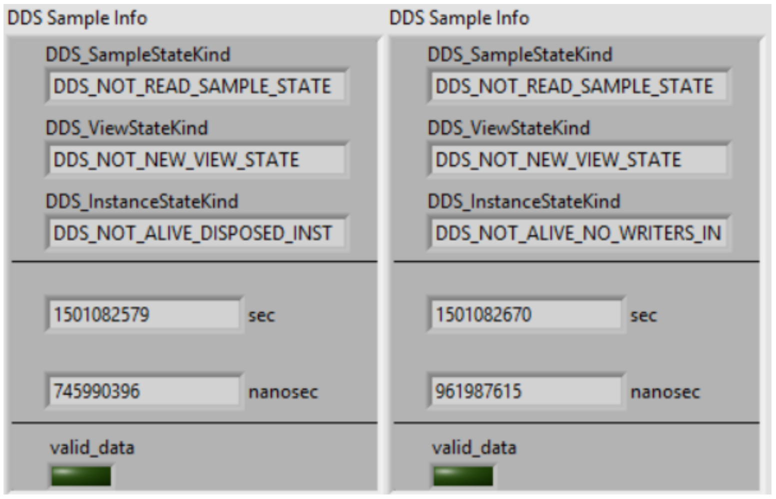
Writing instance state samples (or modifying the instance state of a specific instance) doesn't make sense if you cannot get that information.

The exclusive readers that are not forced to use read() will be able to get samples whose instance state is different than 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.**

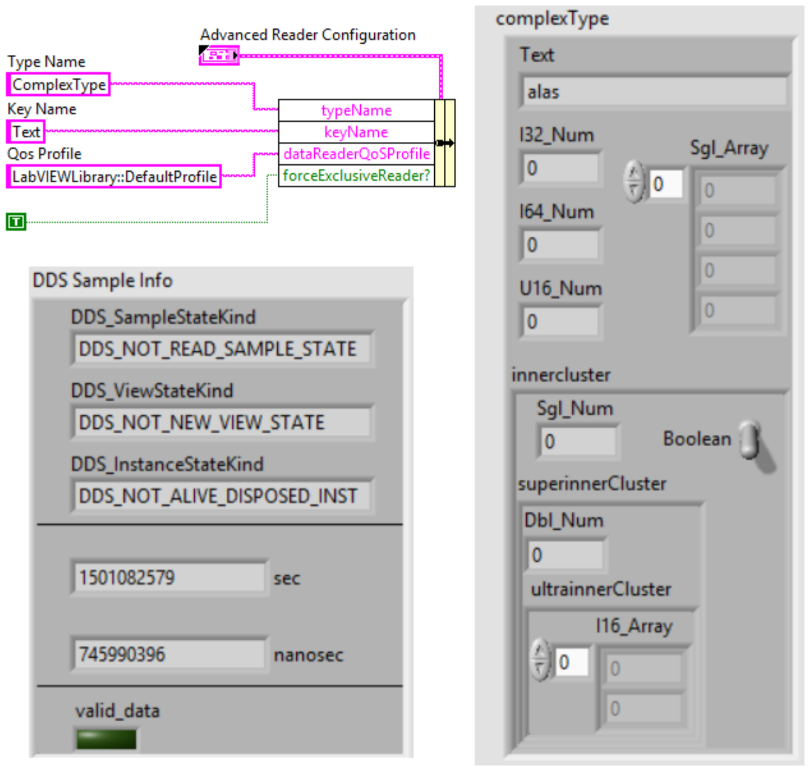
The following pictures shows those ‘dummy’ samples:



DDS Sample Info
DDS_SampleStateKind
DDS_NOT_READ_SAMPLE_STATE
DDS_ViewStateKind
DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind
DDS_NOT_ALIVE_DISPOSED_INST
1501082579 sec
745990396 nanosec
valid_data

DDS Sample Info
DDS_SampleStateKind
DDS_NOT_READ_SAMPLE_STATE
DDS_ViewStateKind
DDS_NOT_NEW_VIEW_STATE
DDS_InstanceStateKind
DDS_NOT_ALIVE_NO_WRITERS_IN
1501082670 sec
961987615 nanosec
valid_data

Using the VIs created in 4.2 Lesson 2—Using ComplexType Generator to Publish and Subscribe to Complex Data (Clusters) on page 38, 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:

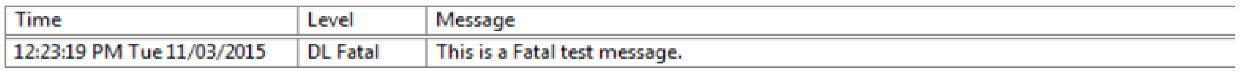
- Fatal
- Severe
- Error
- Warning
- Notice
- Info
- Debug
- Trace
- Silent: This level means that the message will never be stored on the queue.

RTI DDS Toolkit and *RTI Connext* use different filter levels. The following table shows the equivalences:

Table 6.1 Equivalent Filter Levels

RTI DDS Toolkit Filter Level	RTI Connext DDS Core Filter Level
FATAL ERROR SEVERE	ERROR
WARNING	WARNING
NOTICE INFO DEBUG TRACE	STATUS LOCAL

3. Message, which is a string containing useful information.



Time	Level	Message
12:23:19 PM Tue 11/03/2015	DL Fatal	This is a Fatal test message.

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 6.7.3 Logging Messages from LabVIEW on page 119). The queue has two associated 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** (see 6.7.3 Logging Messages from LabVIEW on page 119).

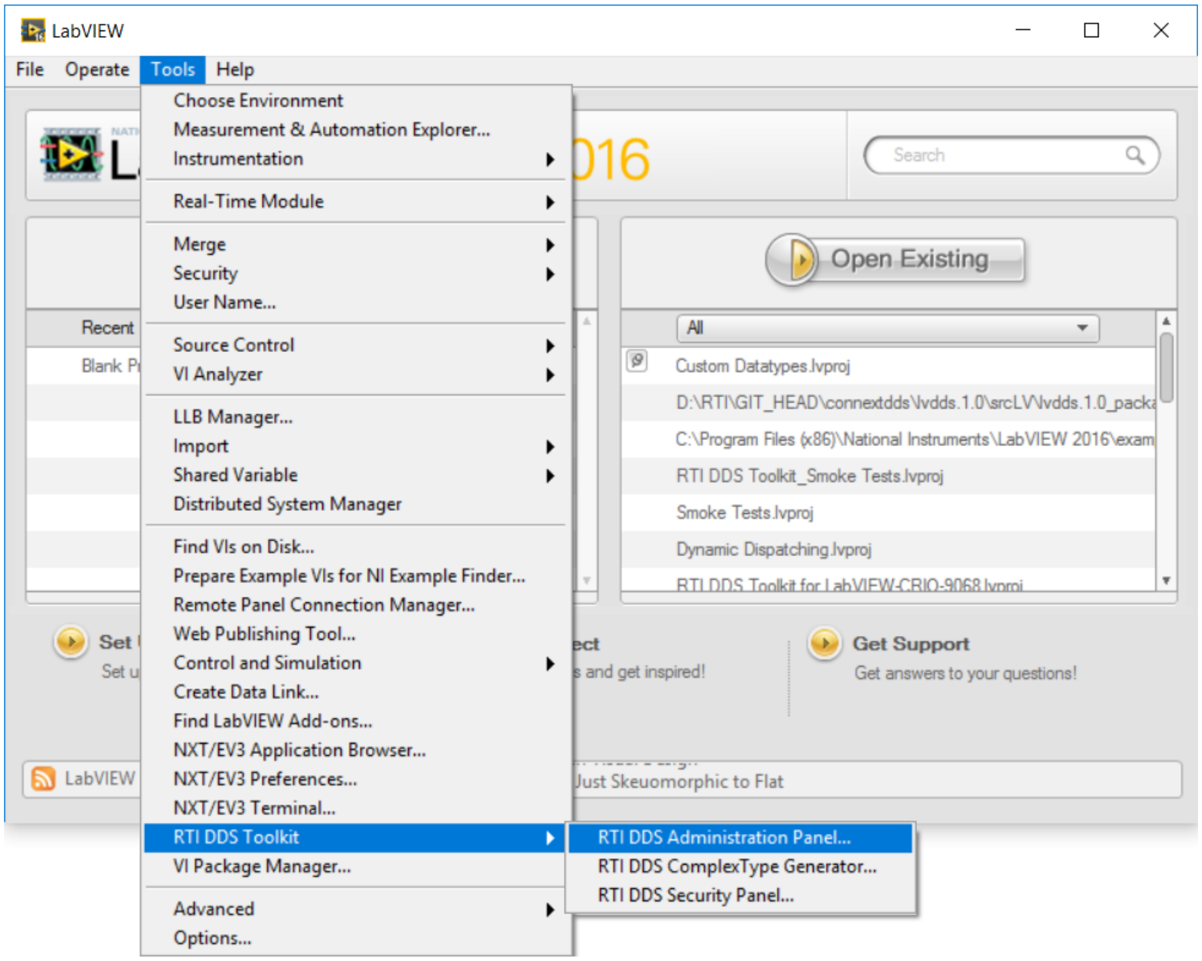
However, once they are in the queue, all messages are treated equally.

6.7.1 Using Administration Panel (for Windows Systems only)

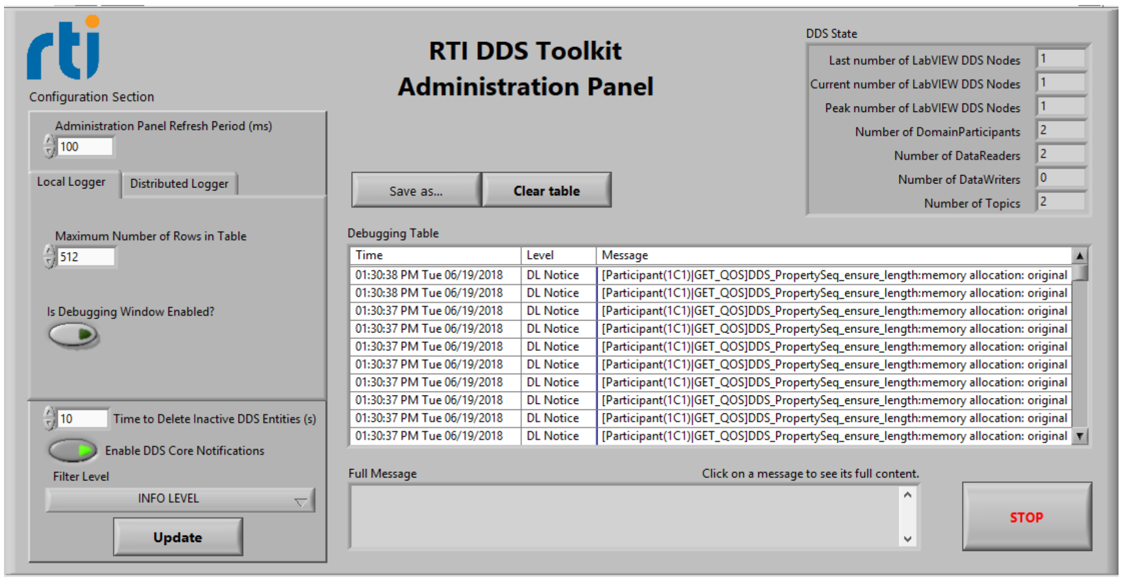
The *RTI DDS Toolkit* Administration Panel is a tool that allows you to administer your DDS applications running on LabVIEW. It also shows diverse DDS information and debugging messages.

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 6.7.2 Debugging SubVIs on Real-Time Targets and Windows Systems on page 116.

You can open the Administration Panel from the Tools menu (**RTI DDS Toolkit, RTI DDS Administration Panel**).


LabVIEW
File
Operate
Tools
Help
Choose Environment
Measurement & Automation Explorer...
Instrumentation
Real-Time Module
Merge
Security
User Name...
Source Control
VI Analyzer
LLB Manager...
Import
Shared Variable
Distributed System Manager
Find VIs on Disk...
Prepare Example VIs for NI Example Finder...
Remote Panel Connection Manager...
Web Publishing Tool...
Control and Simulation
Create Data Link...
Find LabVIEW Add-ons...
NXT/EV3 Application Browser...
NXT/EV3 Preferences...
NXT/EV3 Terminal...
RTI DDS Toolkit
VI Package Manager...
Advanced
Options...
RTI DDS Administration Panel...
RTI DDS ComplexType Generator...
RTI DDS Security Panel...
Search
Open Existing
Recent
Blank P
All
Custom Datatypes.lvproj
D:\RTI\GIT_HEAD\connextdds\lvdds.1.0\srcLV\lvdds.1.0_packa
C:\Program Files (x86)\National Instruments\LabVIEW 2016\exam
RTI DDS Toolkit_Smoke Tests.lvproj
Smoke Tests.lvproj
Dynamic Dispatching.lvproj
RTI DDS Toolkit for LabVIEW-CRIO-9068.lvproj
Set
Set u
ect
s and get inspired!
Get Support
Get answers to your questions!
LabVIEW
Just Skeuomorphic to Flat

Let's take a look at the Administration Panel:



Configuration Section

Administration Panel Refresh Period (ms): 100

Maximum Number of Rows in Table: 512

Is Debugging Window Enabled?

10 Time to Delete Inactive DDS Entities (s)

Enable DDS Core Notifications

Filter Level: INFO LEVEL

DDS State

Last number of LabVIEW DDS Nodes	1
Current number of LabVIEW DDS Nodes	1
Peak number of LabVIEW DDS Nodes	1
Number of DomainParticipants	2
Number of DataReaders	2
Number of DataWriters	0
Number of Topics	2

Debugging Table

Time	Level	Message
01:30:38 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:38 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original
01:30:37 PM Tue 06/19/2018	DL Notice	[Participant(1C1)|GET_QOS]DDS_PropertySeq_ensure_length:memory allocation: original

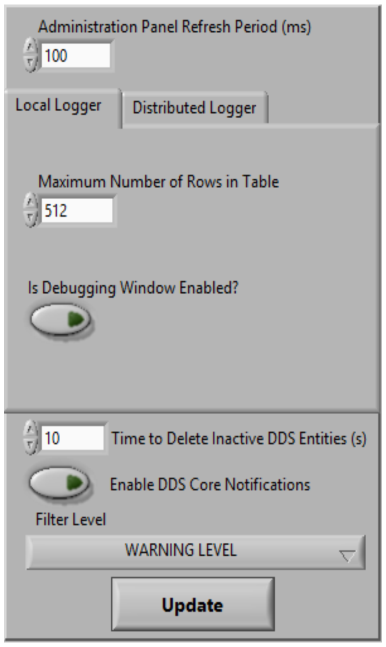
Full Message

Click on a message to see its full content.

- The Configuration section allows you to modify the internal behavior of the toolkit and the Administration Panel itself. See 6.7.1.1 Configuration Section below.
- The DDS state cluster shows information about the internal DDS entities created using *RTI DDS Toolkit*. See 6.7.1.2 DDS State Info on page 114.
- The Debugging table prints the messages stored in the internal logging queue. See 6.7.1.3 Debugging Table on page 115.

6.7.1.1 Configuration Section

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

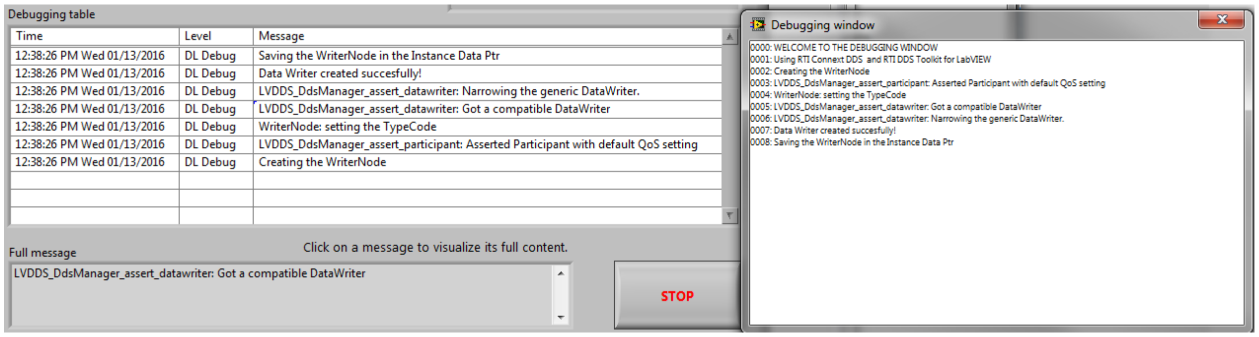

Administration Panel Refresh Period (ms)
100
Local Logger
Distributed Logger
Maximum Number of Rows in Table
512
Is Debugging Window Enabled?
10
Time to Delete Inactive DDS Entities (s)
Enable DDS Core Notifications
Filter Level
WARNING LEVEL
Update

- **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 of 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:


Debugging table

Time	Level	Message
12:38:26 PM Wed 01/13/2016	DL Debug	Saving the WriterNode in the Instance Data Ptr
12:38:26 PM Wed 01/13/2016	DL Debug	Data Writer created succesfully!
12:38:26 PM Wed 01/13/2016	DL Debug	LVDDS_DdsManager_assert_datawriter: Narrowing the generic DataWriter.
12:38:26 PM Wed 01/13/2016	DL Debug	LVDDS_DdsManager_assert_datawriter: Got a compatible DataWriter
12:38:26 PM Wed 01/13/2016	DL Debug	WriterNode: setting the TypeCode
12:38:26 PM Wed 01/13/2016	DL Debug	LVDDS_DdsManager_assert_participant: Asserted Participant with default QoS setting
12:38:26 PM Wed 01/13/2016	DL Debug	Creating the WriterNode

Full message

Click on a message to visualize its full content.

```
LVDDS_DdsManager_assert_datawriter: Got a compatible DataWriter
```

Debugging window

```
0000: WELCOME TO THE DEBUGGING WINDOW
0001: Using RTI Connext DDS  and RTI DDS Toolkit for LabVIEW
0002: Creating the WriterNode
0003: LVDDS_DdsManager_assert_participant: Asserted Participant with default QoS setting
0004: WriterNode: setting the TypeCode
0005: LVDDS_DdsManager_assert_datawriter: Got a compatible DataWriter
0006: LVDDS_DdsManager_assert_datawriter: Narrowing the generic DataWriter.
0007: Data Writer created succesfully!
0008: Saving the WriterNode in the Instance Data Ptr
```

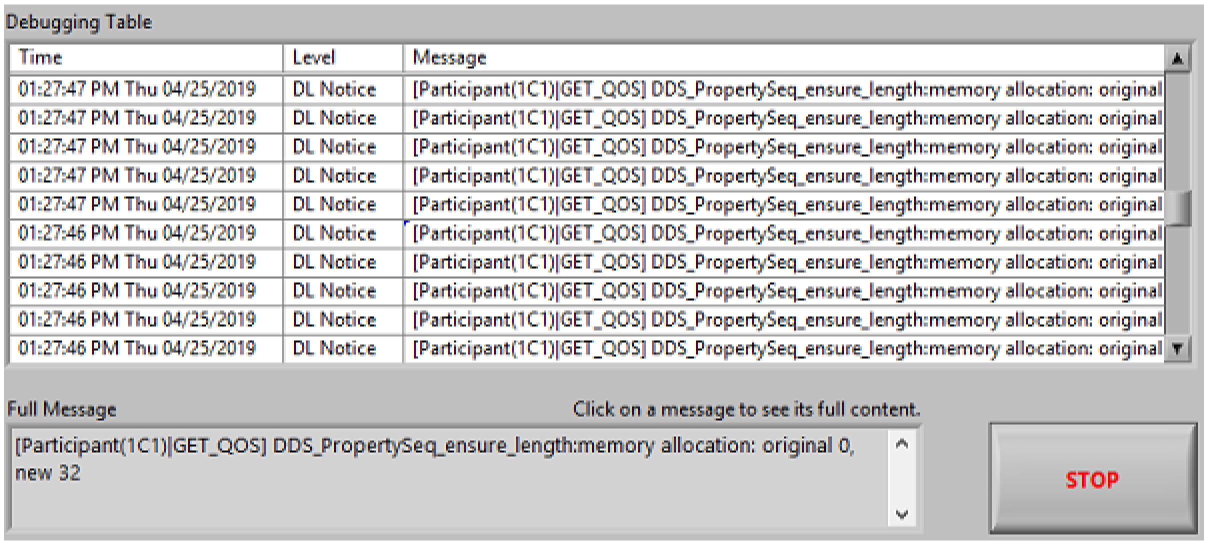
The Debugging window is a tool for printing text information from a LabVIEW application. On Windows systems, the Debugging windows looks like the above figure. However, on NI™ Linux® systems, setting this boolean parameter to True enables messages to be logged to the console out port.

- **Enable DDS Core Notifications:** Enables internal core notifications. These messages refer to internal *Connext* core library logger messages. These messages have the following format:

```
[Entity creating the message | ACTION_IDENTIFIER] Function_name:message
```

Notes:

- Fields in brackets like [Entity creating the message | ACTION_IDENTIFIER] are not always present.
- The filter levels for the *RTI Connext* core and *RTI DDS Toolkit* are different, see Table 6.1 Equivalent Filter Levels on page 109.


Debugging Table

Time	Level	Message
01:27:47 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:47 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:47 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:47 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:47 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:46 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:46 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:46 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:46 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original
01:27:46 PM Thu 04/25/2019	DL Notice	[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original

Full Message

Click on a message to see its full content.

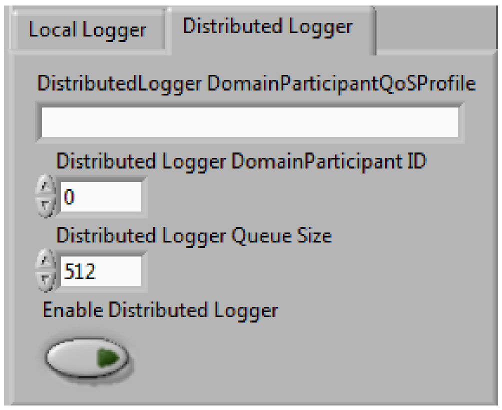
```
[Participant(1C1)|GET_QOS] DDS_PropertySeq_ensure_length:memory allocation: original 0,
new 32
```

- **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.

  Note: You cannot use a Custom Security Profile as the Distributed Logger DomainParticipantQoSProfile. See 6.8.1.1 Creating Custom Security Profiles on page 124.
- **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.

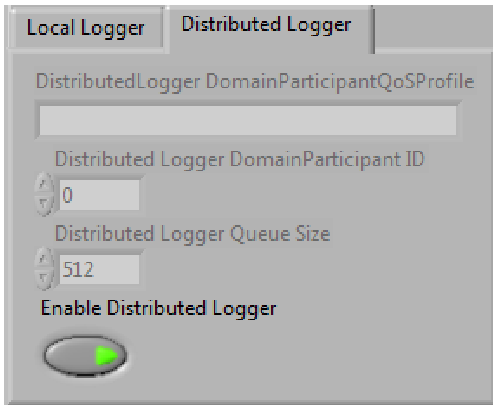


DistributedLogger DomainParticipantQoSProfile

Distributed Logger DomainParticipant ID
0

Distributed Logger Queue Size
512

Enable Distributed Logger



DistributedLogger DomainParticipantQoSProfile

Distributed Logger DomainParticipant ID
0

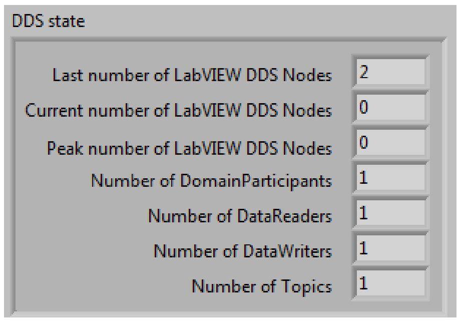
Distributed Logger Queue Size
512

Enable Distributed Logger

- **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. In addition, all DDS entities are deleted when the execution of the last VI that uses *DDS Toolkit* subVIs stops.
- **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 Toolkit*, as well as the internal DDS entities:



DDS state

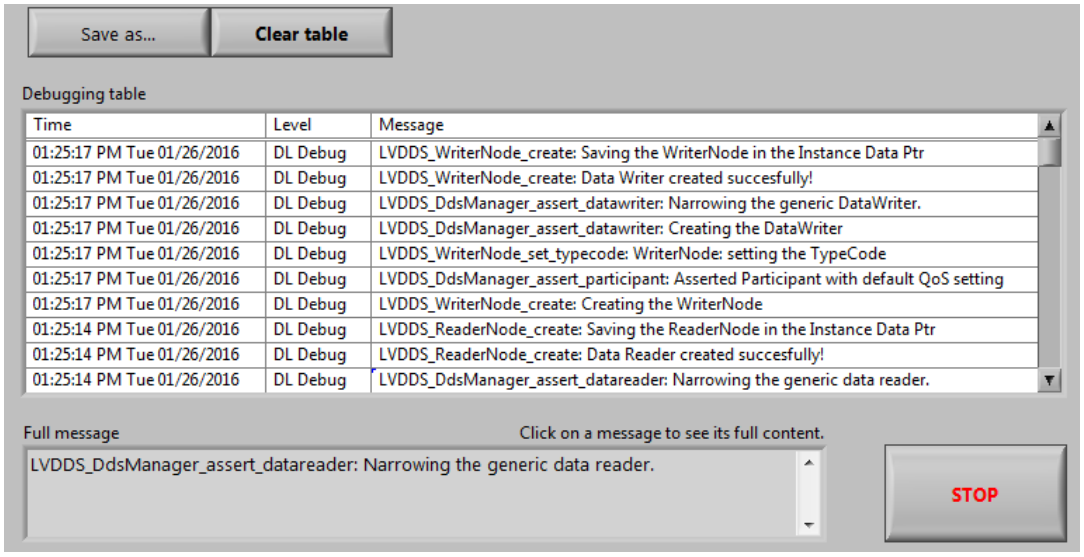
Last number of LabVIEW DDS Nodes	2
Current number of LabVIEW DDS Nodes	0
Peak number of LabVIEW DDS Nodes	0
Number of DomainParticipants	1
Number of DataReaders	1
Number of DataWriters	1
Number of Topics	1

- **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 are 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.



Debugging table

Time	Level	Message
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_WriterNode_create: Saving the WriterNode in the Instance Data Ptr
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_WriterNode_create: Data Writer created succesfully!
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_DdsManager_assert_datawriter: Narrowing the generic DataWriter.
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_DdsManager_assert_datawriter: Creating the DataWriter
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_WriterNode_set_typecode: WriterNode: setting the TypeCode
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_DdsManager_assert_participant: Asserted Participant with default QoS setting
01:25:17 PM Tue 01/26/2016	DL Debug	LVDDS_WriterNode_create: Creating the WriterNode
01:25:14 PM Tue 01/26/2016	DL Debug	LVDDS_ReaderNode_create: Saving the ReaderNode in the Instance Data Ptr
01:25:14 PM Tue 01/26/2016	DL Debug	LVDDS_ReaderNode_create: Data Reader created succesfully!
01:25:14 PM Tue 01/26/2016	DL Debug	LVDDS_DdsManager_assert_datareader: Narrowing the generic data reader.

Full message — Click on a message to see its full content.

LVDDS_DdsManager_assert_datareader: Narrowing the generic data reader.

6.7.2 Debugging SubVIs on Real-Time Targets and Windows Systems

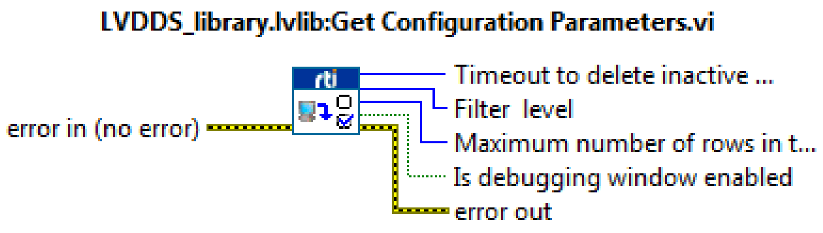
As mentioned in 6.7.1 Using Administration Panel (for Windows Systems only) on page 110, the Administration Panel is not supported on RT Targets. Instead, you can use the following subVIs to debug and administer *RTI Connext DDS* applications deployed on RT targets. These subVIs are in the DDS Debugging 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 6.7.1.1 Configuration Section on page 112:

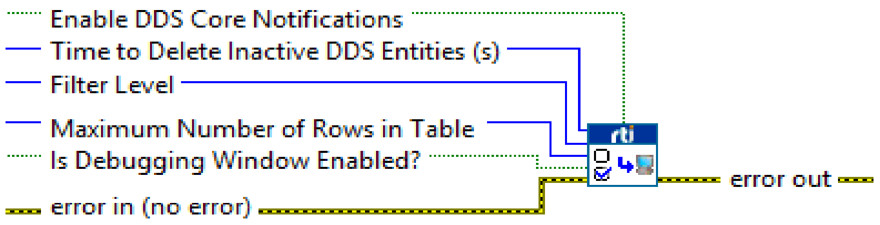
- **Timeout to delete inactive DDS entities**
- **Filter level**
- **Maximum size of the local queue**
- **Is debugging window enabled?**

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


LVDDS_library.lvlib:Get Configuration Parameters.vi
error in (no error)
Timeout to delete inactive ...
Filter level
Maximum number of rows in t...
Is debugging window enabled
error out

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.

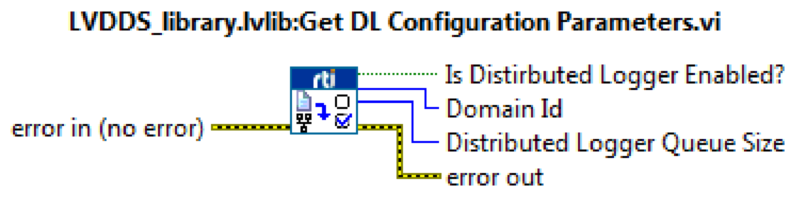

Enable DDS Core Notifications
Time to Delete Inactive DDS Entities (s)
Filter Level
Maximum Number of Rows in Table
Is Debugging Window Enabled?
error in (no error)
error out

6.7.2.3 Get DL Configuration Parameters

This subVI returns the current configuration of the Distributed Logger parameters described in 6.7.1.1 Configuration Section on page 112:

- 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.

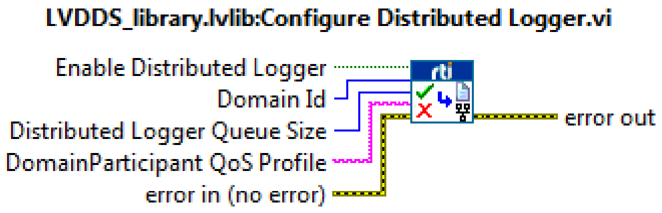

LVDDS_library.lvlib:Get DL Configuration Parameters.vi
error in (no error)
Is Distirbuted Logger Enabled?
Domain Id
Distributed Logger Queue Size
error out

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.

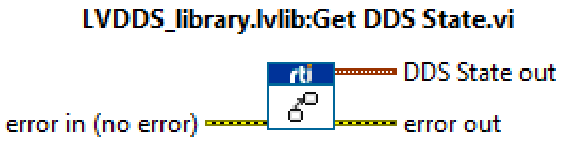
These parameters are used when creating an instance of Distributed Logger:

- **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".


LVDDS_library.lvlib:Configure Distributed Logger.vi
Enable Distributed Logger
Domain Id
Distributed Logger Queue Size
DomainParticipant QoS Profile
error in (no error)
error out

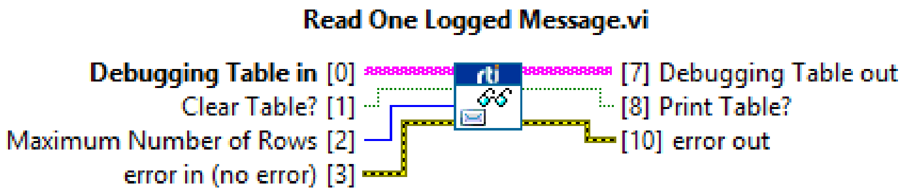
6.7.2.5 DDS State Info

This subVI visualizes the DDS entities created by LabVIEW is controlled by the error wire. The data shown is the same as explained in 6.7.1.2 DDS State Info on page 114.



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".

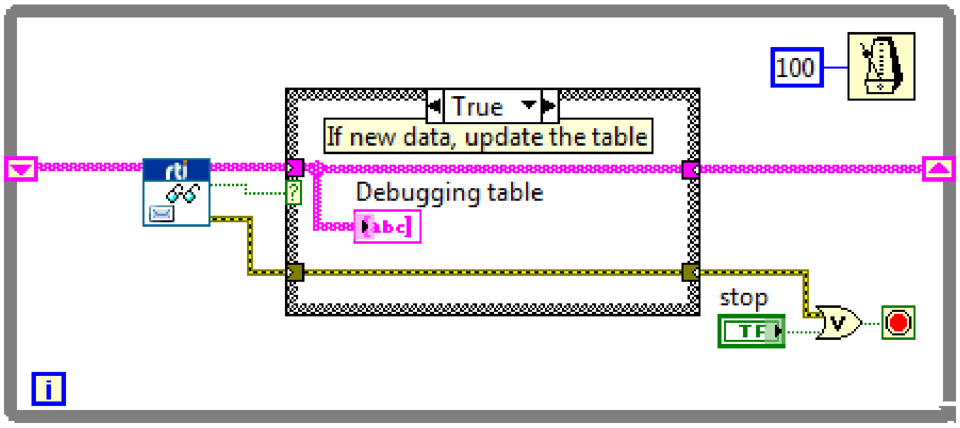


There are pins connected to it:

- 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.



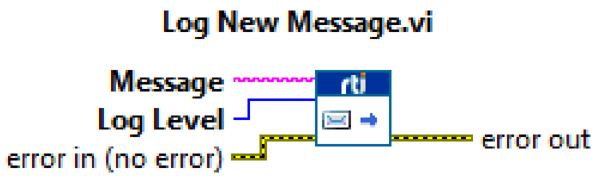
Finally, the flag **Print table?** improves the performance by only updating the table control if a new message was read (or if the table has been cleared).

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

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.
- **Log Level:** The log level with which the message will be registered.



4.9.1.1 Logging Messages Manually on page 74 explains with an example how to use this subVI to log your own messages.

6.8 Enabling Security (Windows only)

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.

Security features are only supported in Windows platforms. If security is enabled in an NI Linux RT target, it will fail with error 5113.

DomainParticipant QoS profiles can be loaded from an XML file, as explained in Chapter 5 Loading Quality of Service Profiles on page 94.

Mandatory Properties:

Unlike previous versions of *RTI DDS Toolkit*, version 3.1.2 now includes OpenSSL version 1.1.1t. The *DDS Toolkit* no longer uses NI SSL. The security library is now embedded into the *DDS Toolkit* library.

At a minimum, your security profile must have the following properties to enable security:

```
<domain_participant_qos>
   <property>
      <value>
         <element>
            <name>com.rti.serv.load_plugin</name>
            <value>com.rti.serv.secure</value>
         </element>
      </value>
   </property>
</domain_participant_qos>
```

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**
- Private Key Decryption Password: password for the private key in case it is encrypted

For further information and latest notes about DDS Security Plugins, see the *RTI Security Plugins Getting Started Guide* and *RTI Security Plugins Release Notes* available here: 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.

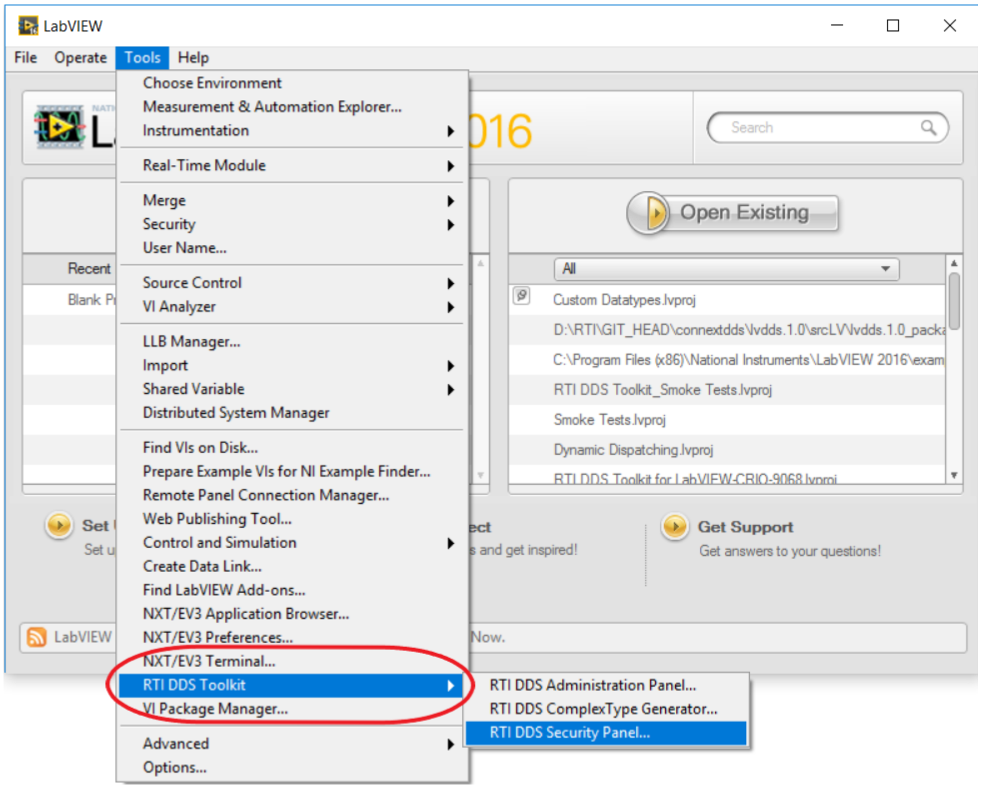
6.8.1 Managing Custom Security Profiles with the Security Panel (Windows only)

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

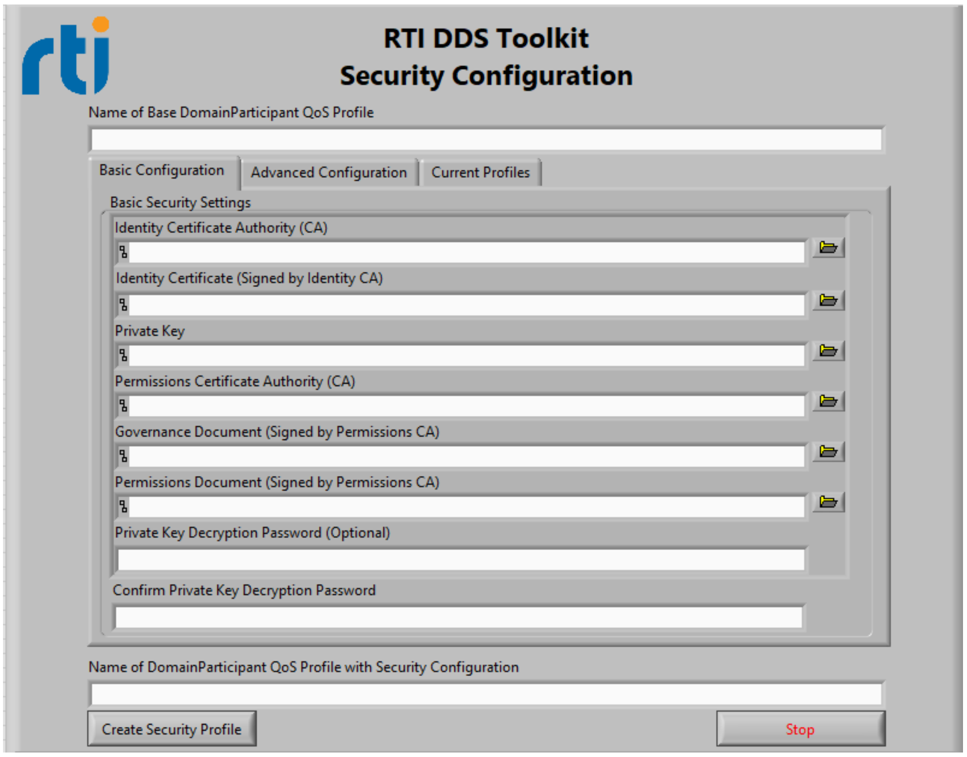
- Create a Custom Security Profile
- Delete a Custom Security Profile
- See the current Custom Security Profiles
- Get **Security Profile Values.vi**

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 6.8.2 Managing Custom Security Profiles with SubVIs on page 125.

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.



RTI DDS Toolkit
Security Configuration

Name of Base DomainParticipant QoS Profile

Basic Configuration | Advanced Configuration | Current Profiles

Basic Security Settings

Identity Certificate Authority (CA)

Identity Certificate (Signed by Identity CA)

Private Key

Permissions Certificate Authority (CA)

Governance Document (Signed by Permissions CA)

Permissions Document (Signed by Permissions CA)

Private Key Decryption Password (Optional)

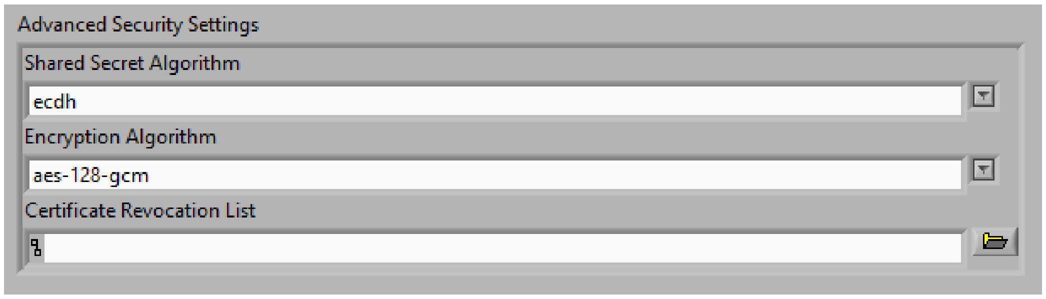
Confirm Private Key Decryption Password

Name of DomainParticipant QoS Profile with Security Configuration

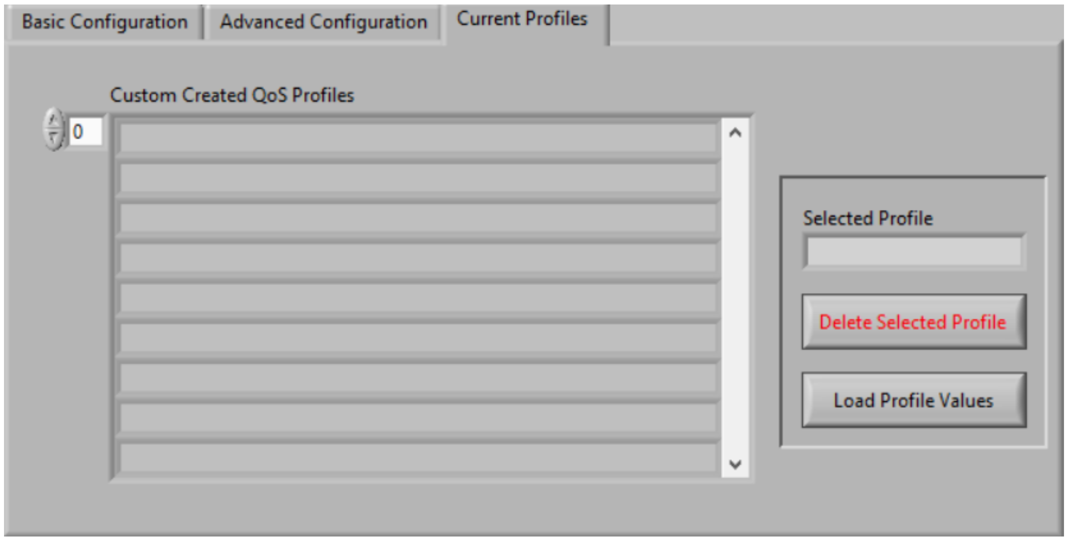
Create Security Profile | Stop

- **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 builtin 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 basic properties noted in 6.8 Enabling Security (Windows only) on page 120.

  If the selected private key is encrypted, you can add a password for it in the profile. Enter the password in the text input boxes ‘Private Key Decryption Password (Optional)’ and ‘Confirm Private Key Decryption Password’. For security reasons, the text will be shown as a sequence of asterisks in both boxes.
- **Advanced Configuration:** Sets the optional properties noted in 6.8 Enabling Security (Windows only) on page 120.


Advanced Security Settings
Shared Secret Algorithm
ecdh
Encryption Algorithm
aes-128-gcm
Certificate Revocation List

- **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 6.8 Enabling Security (Windows only) on page 120, 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 on page 162 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. If the loaded profile uses a password for the private key, it will show in sequence of asterisks in the "Private Key Decryption Password (Optional)". That sequence is a fixed string with a fixed size not related to the password. So the password size CANNOT be inferred from the shown string.

6.8.2 Managing Custom Security Profiles with SubVIs

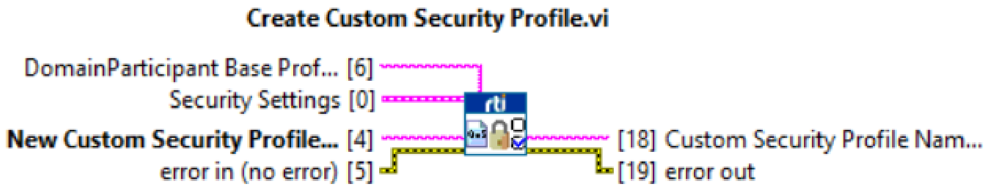
As mentioned in 6.8.1 Managing Custom Security Profiles with the Security Panel (Windows only) on page 121, the Security Panel is not supported on RT targets. Instead, you can use the following subVIs to create your own Custom Security Profiles on these systems.

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


Create Custom Security Profile.vi
DomainParticipant Base Prof... [6]
Security Settings [0]
New Custom Security Profile... [4]
error in (no error) [5]
rti
[18] Custom Security Profile Nam...
[19] error out

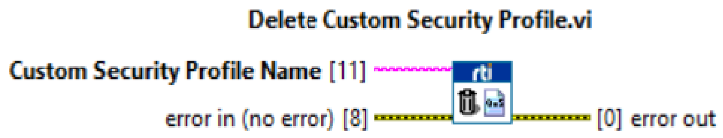
This also returns the Custom Security Profile Name to be used by other subVIs.

These parameters are the same as those described in 6.8.1 Managing Custom Security Profiles with the Security Panel (Windows only) on page 121.

6.8.2.2 Deleting Custom Security Profiles

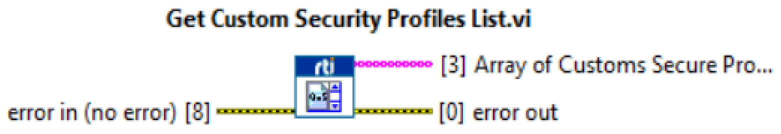
This subVI deletes a Custom Security Profile based on the provided name:

- Custom Security Profile Name which will be deleted.


Delete Custom Security Profile.vi
Custom Security Profile Name [11]
error in (no error) [8]
rti
[0] error out

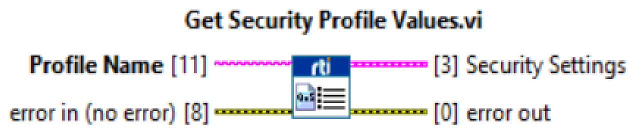
6.8.2.3 Getting Custom Security Profiles List

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


Get Custom Security Profiles List.vi
rti
[3] Array of Customs Secure Pro...
error in (no error) [8]
[0] error out

6.8.2.4 Get Security Profiles Values

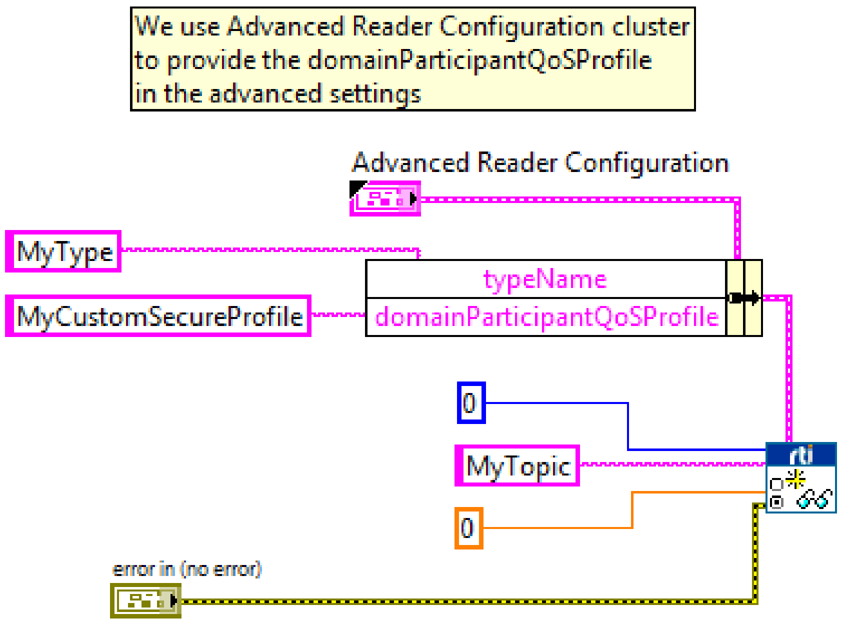
This subVI returns the Security Settings of a specific profile identified by name. The returned Security Settings will contain all the parameters which this Security Profile uses. This means that even if the Security Profile has been created based on another profile that contained any security properties,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



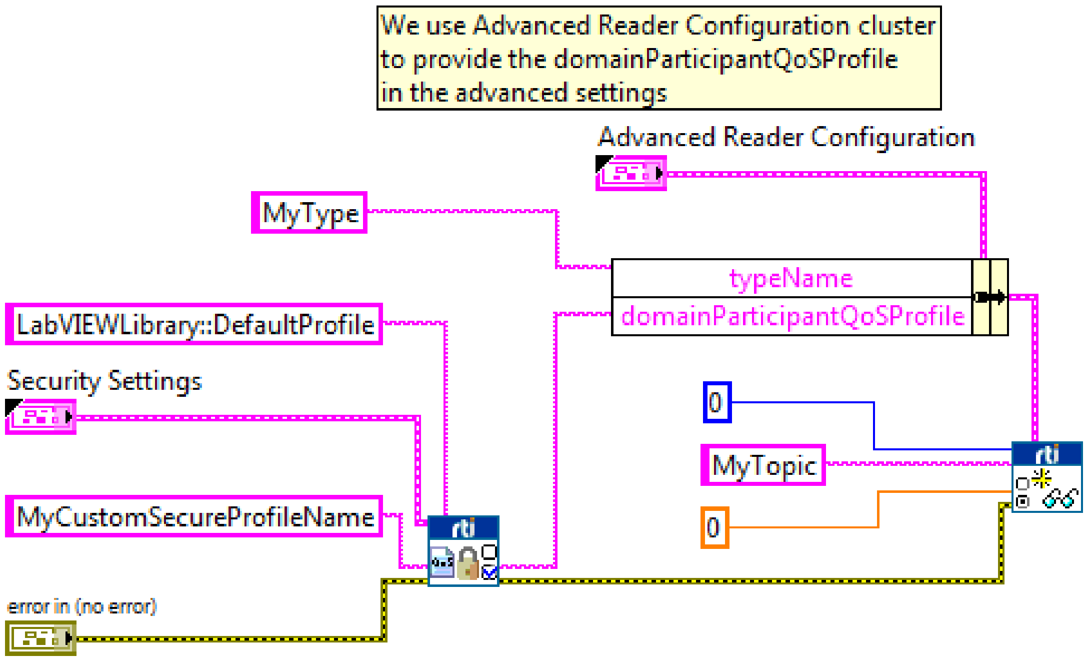
6.8.3 Creating DomainParticipants using a Custom Security Profile

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 6.4 Configuring Advanced Writer Settings on page 103.

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


We use Advanced Reader Configuration cluster to provide the domainParticipantQoSProfile in the advanced settings
Advanced Reader Configuration
MyType
MyCustomSecureProfile
typeName
domainParticipantQoSProfile
0
MyTopic
0
error in (no error)

If the profile has been created using a Custom Security Profile with a subVI:


We use Advanced Reader Configuration cluster to provide the domainParticipantQoSProfile in the advanced settings
Advanced Reader Configuration
MyType
typeName
domainParticipantQoSProfile
LabVIEWLibrary::DefaultProfile
Security Settings
0
MyTopic
MyCustomSecureProfileName
0
error in (no error)

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.

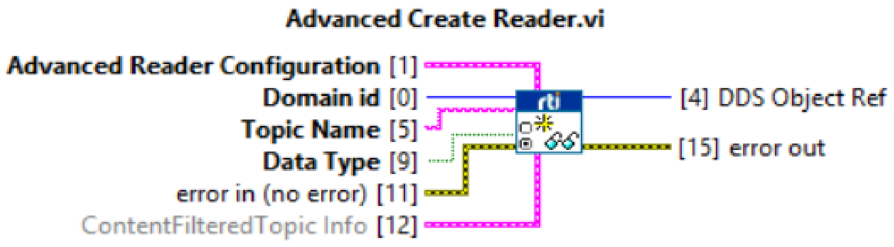
A DataWriter will automatically filter DDS data samples for a DataReader if all of the following are true; otherwise filtering is performed by the DataReader.

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 DomainParticipant of the DataWriter and the DataReader.
7. The DataWriter is not configured to use batching.

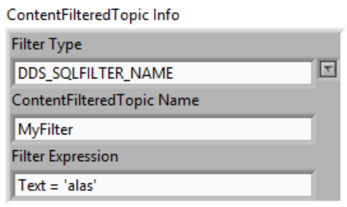
See the *RTI Connext Core Libraries User's Manual* for more details, available here: [https://-community.rti.com/documentation](https://community.rti.com/documentation).

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:



ContentFilteredTopic Info
Filter Type: DDS_SQLFILTER_NAME
ContentFilteredTopic Name: MyFilter
Filter Expression: Text = 'alas'

- **Filter Type:** Filter used to created 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.

6.10 Setting Up Arrays

The length of the arrays in the Data Type is the same as the length in the Type Definition (**.ctl** file). Make sure you declare your array to be the maximum size you will need. To do so, set the array to the desired size and set it as the default value: right-click in the array (not in the contained element) and select **Data Operations, Make Current Value Default**. Do this before saving the Type Definition file and using it with the ComplexType Generator.

Setting up arrays of clusters or strings follows the same rules as native types arrays. The array must be set to the number of members desired and set as default values.

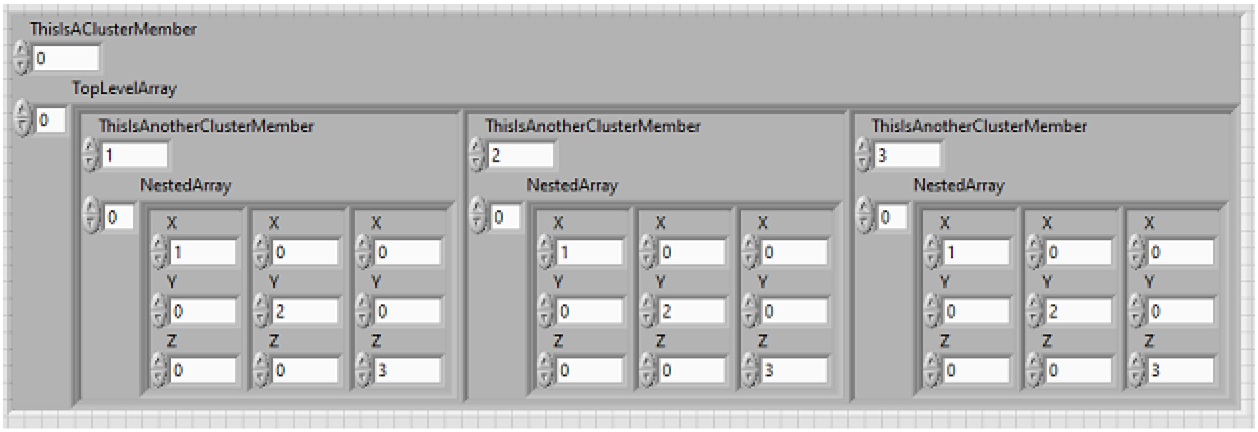
Every LabVIEW one-dimensional array in the Type Definition will be mapped as a DDS Sequence unless the option **forceArrayMapping** is marked in the Advanced Reader/Writer Configuration control. Multi-dimensional arrays will always be mapped as DDS Arrays.

6.10.1 Setting Up Arrays of Clusters

Every array in the Type Definition must be initialized before using it with the *ComplexType Generator*. That includes nested arrays. To avoid doing this manually, start by initializing the arrays of the lower-level cluster and set its values as the default before dragging the cluster into the array holder control.

Then when you initialize the array to its desired size, all members will have their arrays initialized. The same applies to all arrays nested at all levels.

If you have nested arrays inside an array of clusters, those nested arrays must be initialized in each member of the array of clusters. Every array in the entire Type Definition (**.ctl** file) must be initialized to its desired size. To do so, simply initialize the array and make its current value the default before dragging the cluster to the array holder control.

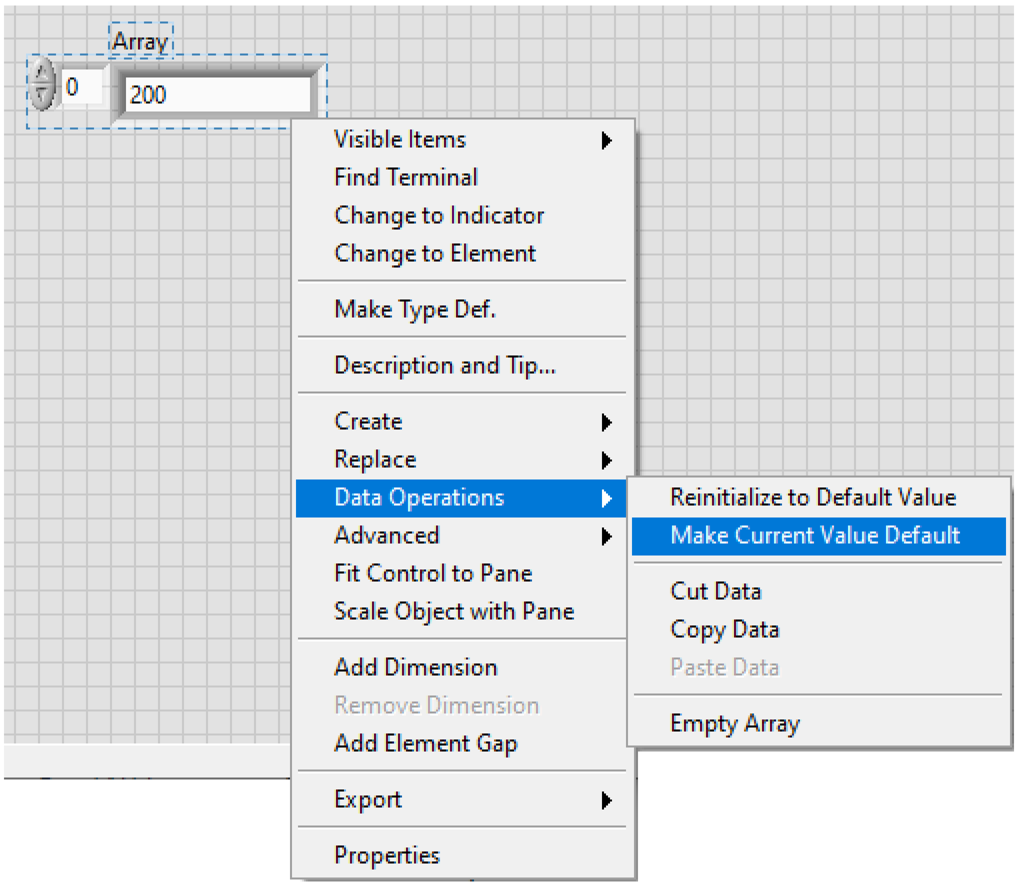


The image above shows how all arrays are initialized. There is a top-level cluster with an array of clusters (**TopLevelArray**); it has three elements. **TopLevelArray** is an array of clusters, each cluster has a nested array of clusters (**NestedArray**). As the image shows, all arrays are initialized. **TopLevelArray** is initialized with size three. Each **NestedArray** inside **TopLevelArray** is also initialized with size three.

6.10.2 Setting up Arrays of Strings

Arrays of strings, like all arrays, must be set to the desired size and each element must be initialized with a default value. To set the size of the string, the first element of the array must contain the size of the string.

For example, to set an array of 50 strings of 200 characters length, initialize the string array with 50 elements. Then in element 0, write “200” and set this as the array's default value: right-click in the array (not the contained string) and select **Data Operations, Make Current Value Default**. Then save the Type Definition (**.ctl** file).



6.10.3 Setting up Sequences

Sequences follow the same rules as arrays. All LabVIEW arrays must be initialized at the sequence's maximum desired size.

By default, every one-dimensional LabVIEW array will be mapped as a DDS Sequence unless the option **forceArrayMapping** in the **Advanced Reader/Writer Configuration** is set. Sequences can only have one dimension. Multi-dimensional LabVIEW arrays will automatically be mapped as DDS Arrays.

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
forceArrayMapping?	Boolean
forceExclusiveReader?	Boolean
forceRead? (only ExclusiveReader)	Boolean
forceUnboundedString?	Boolean

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

Private Key Decryption Password (Optional)	String
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 6.6.1 Write, Dispose or Unregister on page 106 for more information.

DDS Duration: This cluster contains the time period, in seconds and nanoseconds, used in the blocking read operation.

Sec	I32
Nanosec	U32

RTI DDS Read Mode: This enum is used to select the read mode in the *Read* subVI. Values are:

- LVDDS_READ_MODE_POLLING: (Default) Read is performed by polling. If there is no valid data, then return immediately and set the **valid_data** field of the output "Sample Info" to false.
- LVDDS_READ_MODE_BLOCKING: Read keeps waiting until a valid sample can be read or the DDS Duration "Blocking timeout" parameter passed to the *Read* subVI expires.

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:

- A.2.1 Writer on the next page
- A.2.2 Reader on page 137

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.

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 File Folders Installed within LabVIEW on page 153 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.

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)
Read Mode	Enum to set the read operation mode. Can be either: • LVDDS_READ_MODE_POLLING: (Default) Read is performed by polling. If there is no valid data, then return immediately and set the valid_data field of the output “Sample Info” to false. • LVDDS_READ_MODE_BLOCKING: Read keeps waiting until a valid sample can be read or the timeout set in “Blocking Timeout” expires.
Blocking Timeout	Amount of time (seconds and nanoseconds) the Read VI will wait for a valid sample to be read. Default value is 0 seconds and 0 nanoseconds.
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
Timeout	Boolean that indicates if the timeout set in the Blocking Timeout has expired without reading any valid sample. Only used when Read Mode is LVDDS_READ_MODE_BLOCKING.
DDS Sample Info	DDS Sample Info cluster containing information about the sample read.
error out	LabVIEW Error cluster out (optional)

- **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
error in	LabVIEW Error cluster in (optional)

Output parameters

error out	LabVIEW Error cluster out (optional)

- **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 File Folders Installed within LabVIEW on page 153 for details on where this file is located).
error in	LabVIEW Error cluster in (optional)

Output parameters

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

A.2.3 Tools

- **DDS Generate Custom Type VIs:** Generates custom complex types VIs the same way the Complex Type Generator does. For more information, see 6.3 Working with Custom Types on page 99.

Input parameters

Path to the CustomType Definition	Path to the ctl file we want to use.
Output Directory	Where the files will be generated
Type of Generation	Either: • Simple • Advanced
Save the Type Definition	If the type definition must be saved in the output directory
Generate Example VIs	If example files will be created

Domain ID	Domain ID used in the example files
Topic Name	Topic name used in the examples
error in (no error)	Error cluster

Output parameters

Generated Files	Array of paths of the generated files
Error out	Error cluster

- **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

Output parameters

Error Code	*RTI DDS Toolkit* Error Code (optional)
error out	LabVIEW Error cluster out (optional)

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

Input parameters

X	DBL

Output parameters

Time Stamp	Cluster

A.2.3.1 DDS Debugging

- **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

Enable DDS Core Notifications	Boolean - Default: False
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
error in	LabVIEW Error cluster in

  Output parameters

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
error in	LabVIEW Error cluster in

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
error in	LabVIEW Error cluster in

Output parameters

error out	LabVIEW Error cluster out

A.2.4 DDS Security

- **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
error in	LabVIEW Error cluster in

Output parameters

Custom Security Profile Name Output	String
error out	LabVIEW Error cluster out

- **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
error in	LabVIEW Error cluster in

  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
error out	LabVIEW Error cluster out

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

  Input parameters

Profile Name	String
error in	LabVIEW Error cluster in

  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 E.1 Enabling Debugging Mode on page 155.

DDS Entity	Is Created When…	Is Released When…
DomainParticipant	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.	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.*
Topic 'x'	The *Create Writer/Reader* functions are called from LabVIEW and there is not already another valid *Topic.*	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.*
ContentFilteredTopic 'x'	The *Create Advanced Reader* VI is called from LabVIEW with a valid 'ContentFilteredTopic Info' cluster (all the fields have been set).	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.
Subscriber	Never. *RTI DDS Toolkit* uses an implicit subscriber for each *DomainParticipant.*	Never.
Publisher	Never. *RTI DDS Toolkit* uses an implicit publisher for each *DomainParticipant.*	Never.

DDS Entity	Is Created When…	Is Released When…
DataReader for Topic ‘x’	The *Create Reader* function is called and there is not already another valid *DataReader*. If the forceExclusiveReader flag is true in the Advanced Create Reader, a new DataReader is created.	An execution ends and no *DDS Reader* objects have used the *DataReader* within the defined timeout period. The *DDS Release Unused Entities* function is called from LabVIEW and no *DDS Reader* objects are using the DataReader.
DataWriter for Topic ‘x’	The *DDS Create Writer* function is called from LabVIEW and there is not already another valid *DataWriter*.	An execution ends and no *DDS Writer* objects have used the DataWriter within the defined timeout period. The *DDS Release Unused Entities* function is called from LabVIEW and no *DDS Writer* objects are using the DataWriter.

Appendix C Supported Data Types and Corresponding IDL

RTI DDS Toolkit supports these simple and complex data types:

- NUMERIC

INT8[a]	UINT8[a]
INT16	UINT16
INT32	UINT32
INT64	UINT64
FLOAT/SINGLE	
DOUBLE	

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

[a]INT8 and UINT8 are both mapped as octets. We recommend using UINT8, since octets are not signed.

- ARRAYS (OR MULTIDIMENSIONAL[a] ARRAYS) OF TYPE
    - NUMERIC (INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, FLOAT, DOUBLE, CLUSTER, STRING)
    - BOOLEAN
    - ENUM
    - CLUSTERS
    - STRINGS
- 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 Simple Data Types and Corresponding IDL.
- Clusters use a direct mapping of their configuration into a C struct, see C.1 Corresponding IDL for Complex Data Types on page 151.

[a]If you're using multi-dimensional arrays, you must enable the ForceArrayMapping flag. (See 6.4 Configuring Advanced Writer Settings on page 103 and 6.5 Configuring Advanced Reader Settings on page 104 for more information.) By default, *RTI DDS Toolkit* will try to map the array as a sequence, and that can't be done for multi-dimensional arrays.

Table C.1 Simple Data Types and Corresponding IDL

Data Type	Sample Entry in IDL	Default TypeName[a]

INT8
I8 1.23 I8
	`struct Int8Struct{`
	`   octet value;`
	`};`
		DDS_Tiny

INT16
I16 1.23 I16
	`struct Int16Struct{`
	`    short value;`
	`};`
		DDS_Short

INT32
I32 1.23 I32
	`struct Int32Struct{`
	`    long value;`
	`};`
		DDS_Long

INT64
I64 1.23 I64
	`struct Int64Struct{`
	`    long long value;`
	`};`
		DDS_LongLong

UINT8
U8 1.23 U8
	`struct UnsignedInt8Struct{`
	`octet value;`
	`};`
		DDS::Octets

UINT16
U16 1.23 U16
	`struct UnsignedInt16Struct{`
	`    unsigned short value;`
	`};`
		DDS_UnsignedShort

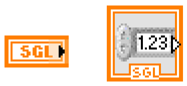
UINT32
U32 1.23 U32
	`struct UnsignedInt32Struct{`
	`    unsigned long value;`
	`};`
		DDS_UnsignedLong

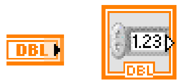
UINT64
U64 1.23 U64
	`struct UnsignedInt64Struct{`
	`    unsigned long long value;`
	`};`
		DDS_UnsignedLongLong

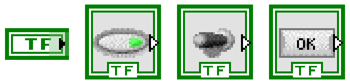
[a]If you do not provide a TypeName, a "Default TypeName" is assigned depending on the type. This may cause conflicts if several cluster types are defined in the same DomainParticipant.

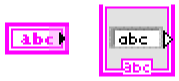
Table C.1 Simple Data Types and Corresponding IDL

Data Type	Sample Entry in IDL	Default TypeName[a]

FLOAT 	`struct FloatStruct{`
	`    float value;`
	`};`
		DDS_Float

DOUBLE 	`struct DoubleStruct{`
	`    double value;`
	`};`
		DDS_Double

BOOLEAN 	`struct BooleanStruct{`
	`    boolean value;`
	`};`
		DDS_Boolean

STRING 	Default:
	`struct DDS_String{`
	`    string<1024> value;`
	`};`
	
	Forcing use of unbounded string:
	`struct DDS_String{`
	`    string value;`
	`};`
		DDS::String

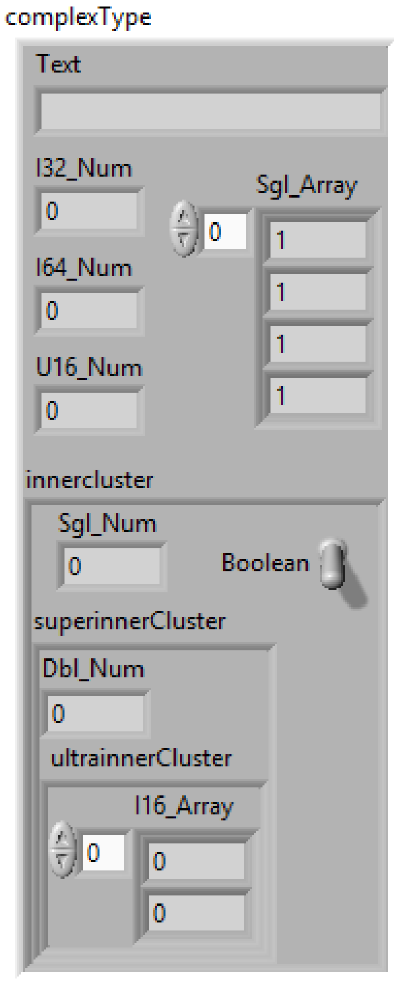
ARRAY of the above types
(This example uses INT16 and nDim elements.)
	Default:
	`struct ArrayStruct {`
	`    sequence<short, nDim> value;`
	`}`
	
	Forcing use of array:
	`struct ArrayStruct`
	`{`
	`    short value[nDim];`
	`}`
		DDS_Default_TypeName

[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:

```
struct MyTypeName{
    string<1024>[1] 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;
};
```

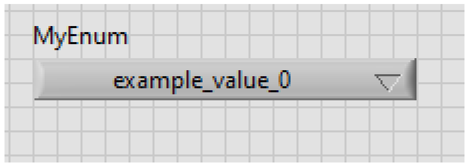
Note that inner clusters add “Type” to their name to avoid repeating the same name in both type and member. Also note that all the names of the components are joined by underscores instead of using spaces. This prevents compiling errors in other languages such as C, C++, Java or .Net. Please consider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

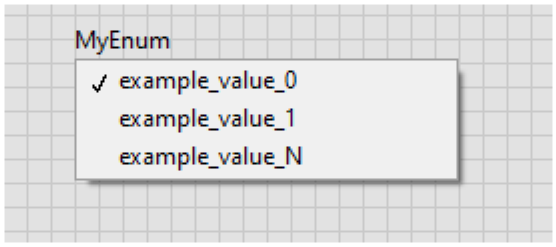
[1] If **forceUnboundedString?** is set to **true**, IDL correspondence will be *string Text;. And* you will need to run the rtiddsgen with the option –unboundedSupport.

C.1.2 Enums

The IDL representation for an enum depends on the elements it is composed of. Note that only 32-bit enums are supported. Enums are represented in LabVIEW controls as Rings or Enums. Those Rings or Enums must have "**allow undefined values in runtime**" disabled, and a representation of U32 for Enums or I32 for Rings.

For example, consider the enum seen below:





It would have the following IDL representation for a Type Name, **MyType**:

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

Note: *DDS Toolkit* will add the suffix "Enum" to the enum name in the Data Type. So if your type is **MyEnum** as seen above, the resulting name in the Data Type will be **MyEnumEnum**.

When updating from a version before 3.1.0: See the Compatibility section of the Release Notes for details on how to replace an enum with a Ring.

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[1]\National Instruments\LabVIEW 20xx**

Where *xx* represents the LabVIEW version number (LabVIEW 2016, 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\ArrayOfClustersDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\BlockingReadDemo
  - \LabVIEW 20xx\examples\RTI DDS Toolkit\ClusterDemo

[1]On 64-bit systems, the folder is "Program Files (x86)"

- \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\ReadMultipleSamplesDemo
- \LabVIEW 20xx\examples\RTI DDS Toolkit\SecurityShapesDemo
- \LabVIEW 20xx\examples\RTI DDS Toolkit\ShapesDemo
- \LabVIEW 20xx\examples\RTI DDS Toolkit\StringsDemo

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 6.7 Debugging an RTI Connext DDS LabVIEW Application on page 108.

E.2 Error Codes and Possible Solutions

Table E.1 Error Codes below shows error codes and possible solutions.

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5001	Something failed in a previous stage (wired error input)	*RTI DDS Toolkit* found an error status in the input error cluster. It might be due to an error in the previous stage.	
5003	Unable to delete the contained entities of a participant	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**)
5004	Unable to delete a participant	It is likely that another application is still using that Participant. Close all the instances before trying to delete it.	
5005	Unable to finalize the DomainParticipantFactory.	It is likely that another application is still using the DomainParticipantFactory. Close all the instances before trying to delete it.	
5006	Bad QoS settings	QoS setting format is incorrect or does not match with any of the ones existing in the XML file or Custom Security Profiles. Check that the corresponding QoS profile exists as a Custom Security Profile or that it is a normal QoS Profile with a correct format (**Library::Profile**). If loading the profile from an XML file, check that the XML file exists and contains the QoS profile with a correct configuration.	

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5007	Unable to assert (find or create) a Participant.	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.	Review the QoS profile for the Participant. Modify the QoS profile to work without an active network interface as explained in E.3 Running without an Active Network Interface on page 165.
5008	Unable to register the type because there exists another entity with same configuration	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.	
5009	Unable to get the Participant QoS for a given profile.	Possible error in the QoS configuration. Check that the corresponding QoS profile exists as a Custom Security Profile or that it is a normal QoS Profile with a correct format (**Library::Profile**). If loading the profile from an XML file, check that the XML file exists and contains the QoS profile with a correct configuration. You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.	Review the QoS profile for the Participant. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**).
5010	Unable to update the number of applications accessing to the Participant (client count property).	This might cause a memory leak when releasing the participant.	
5011	Unable to set the QoS Properties to the participant.	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.	Review the QoS profile for the Participant.
5012	Unable to get the description of the topic.	Check that the *Reader/Writer* was correctly created (no previous errors).	
5014	Unable to assert (find or create) a Topic.	Possible error in the QoS configuration. Check that the corresponding QoS profile exists as a Custom Security Profile or that it is a normal QoS Profile with a correct format (**Library::Profile**). If loading the profile from an XML file, check that the XML file exists and contains the QoS profile with a correct configuration. You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.	Review the QoS profile for the Topic. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**).

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5015	Unable to get the implicit publisher.	Implicit publisher is needed to create the Writer. Check that the participant configuration is correct and that there are no previous errors.	Review the QoS profile for the Publisher. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**). You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.
5016	Unable to get all the DataWriters in the given participant.	It might be due to a memory restriction (not enough memory available to recover the existing DataWriters). Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this problem.	
5017	Unable to create the DataWriter.	Check that the QoS configuration provided for the DataWriter is correct.	Review the QoS profile for the DataWriter. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**). You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.
5018	Unable to get the QoS Properties from a DataWriter.	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).	
5019	Unable to set the QoS Properties for a DataWriter.	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).	
5020	Unable to update the number of applications using a DataWriter.	This might cause a memory leak when releasing the DataWriter.	
5021	Unable to narrow the Dynamic DataWriter.	This is an unexpected error. Contact **labview@rti.com** or visit our Community Portal at http://community.rti.com to view current solutions and forum entries.	
5022	Unable to get the implicit subscriber.	Implicit subscriber is needed to create the Reader. Check that the participant configuration is correct and that there are no previous errors.	Review the QoS profile for the Subscriber. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**). You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5023	Unable to get all the DataWriters in the given participant.	It might be due to a memory restriction (not enough memory available to recover the existing DataWriters). Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5024	Unable to create the DataReader.	Check that the QoS configuration provided for the DataReader is correct.	Review the QoS profile for the DataReader. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**). You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.
5025	Unable to get the QoS Properties from a DataReader.	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).	
5026	Unable to set the QoS Properties for the DataReader.		
5027	Unable to update the number of applications using a DataReader.	This might cause a memory leak when releasing the DataReader.	
5028	Unable to narrow the Dynamic DataWriter.	This is an unexpected error. Contact **labview@rti.com** or visit our Community Portal at http://community.rti.com to view current solutions and forum entries.	
5029	Unable to delete a Topic.	It is likely that another instance of LabVIEW is still using that Topic. Close all LabVIEW instances before trying to delete it.	You can also delete the unused contained entities by using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**).
5030	Unable to delete a DataReader (or its contained entities).	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.	
5031	Unable to delete a DataWriter (or its contained entities).	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.	
5032	Unable to initialize the DDS Dynamic Data.	There was a problem when allocating memory. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5033	Unable to initialize the Reader Node.	There was a problem when allocating memory. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5034	Unable to initialize the DDS Manager.	Check that the DLL was correctly loaded (a message can be found in the Debug Window).	
5035	Invalid reference to a Reader or Writer Node.	Please use the appropriate *Create* subVI to generate a correct reference and connect it to the *Read/Write* subVI.	Pay special attention to the data type.
5036	Unable to read data from DataReader.	Check that the Query Condition is correctly set.	* will return everything. A regular expression will also work (for instance: Text='hello').

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5037	Unable to initialize the Writer Node.	There was a problem when allocating memory. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5038	Unable to write data.	DataWriter timed out or ran out of resources. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this. Check that you attached a valid indicator/storage to the write output.	
5039	Unable to initialize the semaphore for the DLL.	There was a problem when allocating memory. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5040	Unable to create the Query Condition to filter Read subVI.	Check that the Query Condition is correctly set. To read everything, set it to ***** or leave it empty.	A regular expression will also work (for instance: Text-='hello').
5042	Unable to unregister the Type Code.	Other applications might be using it. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5044	Unable to get all the available Topics.	It might be due to a memory restriction (not enough memory available to recover the existing Topics). Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5045	Warning: Unable to delete one or several DDS Entities.	Other instances of LabVIEW are currently using one or several of the DDS Entities.	This is not an error, just a warning. Closing all running VIs should release all the remaining DDS Entities.
5046	Unable to get the Topic's QoS.	Check that the Topic's QoS provided was correct and that the Topic was initialized using the *Create Reader* or *Create Writer* subVI.	Review the QoS profile for the Topic. Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**). You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.
5047	Unable to set the Topic's QoS.		
5048	Unable to access library handler.	Possible error in the QoS properties provided. The RTI DDS Toolkit Dynamic Library was not correctly loaded.	
5049	Unable to take the semaphore	Another thread may already be using the DLL.	

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5050	Unable to recover participant's default QoS	Internal error due to default configuration issues. Contact **labview@rti.com** or visit our Community Portal at http://community.rti.com to view current solutions and forum entries.	
5051	Unable to load QoS profiles from the embedded configuration or external XML files	Error in QoS properties. Verify all profiles loaded by the **NDDS_QOS_PROFILES** environment variable.	Make sure you are selecting the correct settings: either a Custom Security Profile or a fully qualified name (**Library::Profile**). You can also use the default configuration by setting the QoS fields in the Advanced Writer/Reader Configuration cluster (in the Advanced *Create Reader/Writer* subVI) to empty strings.
5052	Incorrect type name.	Usual format is **Library::Type**. Avoid using spaces.	
5053	One of the required parameters of the subVI is missing	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**.	These pins are also required for the clusters even if you use Call Library Function (CLF) calls instead of a subVI.
5054	Unable to access to the Type Code Factory.	Another application has finalized the TypeCode Factory and there was an error while reinitializing it. Retry.	
5056	Unable to create the Type Code.	The attached cluster is incompatible with the supported one and cannot be created.	See Appendix C Supported Data Types and Corresponding IDL on page 147 for details on the supported types.
5057	Unable to set the Dynamic Data.	Check that the correct data type is connected to the subVI (pay special attention to *Create Reader/Writer* ones).	
5058	Unable to get the Dynamic Data.	Check that the correct data type is connected to the subVI (pay special attention to *Read/Write* ones).	
5059	Invalid profile provided to the Set QoS subVI.	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.	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**).
5060	Unable to give the semaphore.		This might block another thread from using the *RTI DDS Toolkit* API.

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5061	Unable to lock/unlock the Participant to create the Reader.	Another application was already deleting the Participant. Removing unused entities or closing the VIs might fix this problem.	You can also delete the unused contained entities by using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit Tools**).
5062	Reached the maximum number of participants allowed in the system.	Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5063	Unable to create the system clock.	This is an unexpected error. Contact **labview@rti.com** or visit our Community Portal at http://community.rti.com to view current solutions and forum entries.	
5064	Unable to create the Type Support needed to register a type.	There was a problem when allocating memory. Using the *DDS Release Unused Entities* subVI (in **RTI DDS Toolkit, Tools**) might fix this.	
5065	Unable to assign that type name to the Topic because it is currently in use.	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.	
5067	Unable to create the key with the provided string. Might be a memory allocation problem.	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 4.7 Lesson 7–Used Nested and Multiple Keys on page 61 for further details.
5068	Unable to create DataReader using read() with KEEP_ALL history kind. Use case not supported.	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.
5069	Incompatible configuration: History depth > 1 needs 'only_new_samples' flag in the Read subVI to be 'true'.	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.
5070	Unable to extract information from the Advanced Writer Configuration control.	Make sure you are using the cluster 'RTI DDS Advanced Writer Configuration.ctl' contained in LVDDS_Library.	
5071	Unable to extract information from the Advanced Reader Configuration control.	Make sure you are using the cluster 'RTI DDS Advanced Reader Configuration.ctl' contained in LVDDS_Library.	
5072	The Local Logger is not correctly initialized.	Make sure the size of the Local Logger is not a negative number.	
5073	Unable to create a new message into the Local Logger.	Make sure there is enough memory to log a new message. You could need to use a lower queue size.	
5074	Unable to create Distributed Logger.	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.	

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5075	Unable to delete Distributed Logger.	Make sure Distributed Logger has not been previously deleted.	
5076	Unable to create the custom QoS list. Might be a memory allocation problem.	The Custom QoS Security Profiles list has not been able to be allocated. Free memory and try again.	
5077	Unable to create the custom QoS Profile. The name might be in use or the input parameters are not correctly set	The Custom QoS Profile cannot be created. Make sure the provided name is not already in use, does not contain whitespaces and the input parameters are correctly set. Check that those parameters are valid paths and do not include the prefixes "file:" or "data:".	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.
5078	Unable to delete the custom QoS Profile. The custom profile might not exist.	The Custom QoS Profile cannot be deleted. Make sure the provided name is created and doesn't contain whitespaces.	The Custom Security Profile you want to delete contains whitespaces or it hasn't been created yet.
5079	Unable to allocate memory for showing the created custom QoS profiles.	Unable to allocate memory for showing the created custom QoS profiles.	
5080	Unable to assert (find or create) a Secure Participant.	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.	
5081	Unable to assert (find or create) a Secure Topic.	Check that the QoS profile exists and the TopicName is allowed by the Security Permissions.	
5082	Unable to assert (find or create) a Secure DataReader.	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.	
5083	Unable to assert (find or create) a Secure DataWriter.	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.	
5084	Unable to load the custom QoS Profile. The custom profile might not exist.	The Custom QoS Profile cannot be loaded. Make sure the provided name exists and does not contain whitespaces.	The Custom Security Profile Name already exists or contains whitespaces.
5085	Unable to assert (find or create) a ContentFilteredTopic	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.	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.
5086	Unable to delete ContentFilteredTopic.	Unable to delete a ContentFilteredTopic. Check that no DataReaders are using it.	The ContentFilteredTopic may not exist or any DataReader is still using it.

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5087	Unable to get all the available ContentFilteredTopics	It might be due to a memory restriction (not enough memory available to recover the existing ContentFilteredTopics). Using the *Release Unused Entities* subVI (in RTI DDS Toolkit, Tools) might fix this.	Try to run the *Release Unused Entities* subVI manually. This subVI is under Data Communication, RTI DDS Toolkit, Tools.
5088	Unable to modify the ContentFilteredTopic because it is currently in use.	The ContentFilteredTopic cannot be deleted because a DataReader is still using it. The Filter Expression of a ContentFilteredTopic cannot be modified while a DataReader is using it.	A ContentFilteredTopic already exists with that name and a different Filter Expression. It cannot be modified.
5089	Expecting a Reader, got a Writer.	The DDS Object Ref that is being used is not a Reader. Make sure that this DDS Object Ref as been created by a *Create Advanced/Simple Reader.*	
5090	Expecting a Writer, got a Reader.	The DDS Object Ref that is being used is not a Writer. Make sure that this DDS Object Ref has been created by a *Create Advanced/Simple Writer*.	
5091	Unable to dispose an Instance.	Error disposing an Instance. Check that the key of the provided data is correctly set.	
5092	Unable to unregister an Instance.	Error unregistering an Instance. Check that the key of the provided data is correctly set.	
5093	Error creating type info list. This might be caused by running out of memory.	Not enough memory for allocating a Type Info List.	
5094	Unable to delete Type Code. This might be caused by a corrupted Type Code due to malformed data. Check Type Definition for inconsistencies.	Corrupted Type Code. This might happen when the Type Code was not created properly.	
5095	Error asserting Type Info. This might happen when a Type Info doesn't exist or another one with the same name already exists but represents a different Type Definition.	The type info doesn't exist or there is another type with the same name but different type or configuration like using arrays instead or sequences.	Try changing the type name of the conflicting type.
5096	Error when creating Type Info. Check the Administration Panel to see more details about where the error happened.	An error occurred while processing a member of the Type Definition. Check the administration panel to see what the conflicting member is.	
5097	Error when reading from LabVIEW data. If using "forceArrayMapping?" option in the Advanced Reader/Writer Configuration, check that arrays have the same size as the Type Definition. If using sequences or strings, check that the length is smaller than the maximum.	An error occurred while reading the cluster members. This might happen if the size of the arrays, sequences or strings passed to the write VI has an invalid size (for example, a sequence with more elements than the maximum set, or an array with a different number of elements than the one in the Type Definition when the VIs were generated).	
5098	Error when writing to LabVIEW data. This may happen when running out of memory or when an invalid address is found. Check the Administration Panel for more detailed information.	An error occurred while trying to write into the output cluster. This might happen because an invalid memory address has been found.	

Table E.1 Error Codes

Error Code	Error Message	Possible Reason(s)	Additional Information
5099	Unable to delete element from the Type Info list. Check the Administration Panel for more details.	An error occurred while trying to delete an existing Type Info. The Administration Panel may provide more information.	
5101	Unable to add TypeInfo node to TypeInfo list.		
5102	Error when adding LabVIEW Cluster to Dynamic Data.	Error when trying to add a cluster to a Dynamic Data struct for sending through a DataWriter.	
5103	Error when adding LabVIEW Array to Dynamic Data array.	Error when trying to add a LabVIEW array to a Dynamic Data struct for sending through a DataWriter.	
5104	Error when adding LabVIEW Array to Dynamic Data sequence.	Error when trying to add a LabVIEW array to a Dynamic Data sequence for sending through a DataWriter	
5105	Error when adding LabVIEW String to Dynamic Data.	Error when trying to add a LabVIEW string to a Dynamic Data struct for sending through a DataWriter	
5106	Error when adding LabVIEW Numeric to Dynamic Data.	Error when trying to add a LabVIEW numeric element to a Dynamic Data struct for sending through a DataWriter.	
5107	Error when adding Dynamic Data struct to LabVIEW Cluster.	Error when converting a Dynamic Data struct into a LabVIEW cluster.	
5108	Error when adding Dynamic Data array to LabVIEW Array.	Error when converting a Dynamic Data array into a LabVIEW array.	
5109	Error when adding Dynamic Data sequence to LabVIEW Array.	Error when converting a Dynamic Data sequence into a LabVIEW array.	
5110	Error when adding Dynamic Data string to LabVIEW String.	Error when converting a Dynamic Data string into a LabVIEW string.	
5111	Error when adding Dynamic numeric to LabVIEW Numeric.	Error when converting a Dynamic Data numeric type into a LabVIEW numeric type.	
5112	Error when adding data to DDS sequence. The number of elements exceeds the maximum allowed.	Check the size of the array before passing it to the Write VI.	
5113	Security is not available in RT targets. Disable security when using RTI DDS LabVIEW in an RT target.	Check the QoS. Security is enabled when running the Toolkit in NI Linus RT targets, but it is not supported. Do not use security in NI Linux targets.	

E.3 Running without an Active Network Interface

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:

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

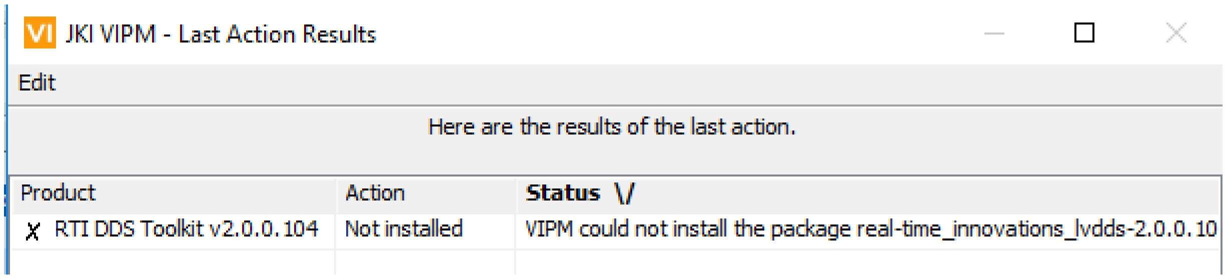
- Another option is to install the Microsoft KM-TEST Loopback Adapter, which simulates the existence of a network interface. You can install it from the Windows Device manager.

  For example, to install the Microsoft KM-TEST Loopback Adapter on a Windows 10 system:

  1. Right-click the Windows Start button and select **Device manager**.
  2. From the Action menu, select **Add legacy hardware**. (If you only see **Help** under the Action menu, select something in the tree first, then look again.)
  3. The Add Hardware Wizard will open. Click **Next**.
  4. Choose **Install the hardware that I manually select from a list (Advanced)** and click **Next**.
  5. Scroll down to select **Network adapters** and click **Next**.
  6. Select **Microsoft** as the manufacturer, select **Microsoft KM-TEST Loopback Adapter** as the model, then click **Next**.
  7. Click **Next**, then **Finish**.

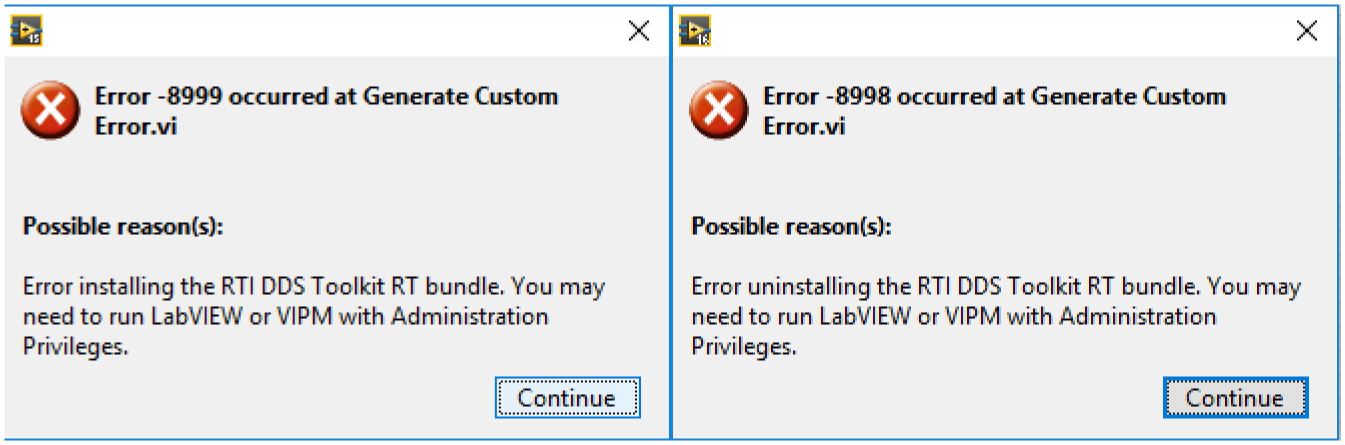
E.4 Error Installing RTI DDS Toolkit RT Support

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.



Here are the results of the last action.

Product	Action	Status
RTI DDS Toolkit v2.0.0.104	Not installed	VIPM could not install the package real-time_innovations_lvdds-2.0.0.10


Error -8999 occurred at Generate Custom Error.vi

Possible reason(s):

Error installing the RTI DDS Toolkit RT bundle. You may need to run LabVIEW or VIPM with Administration Privileges.

Error -8998 occurred at Generate Custom Error.vi

Possible reason(s):

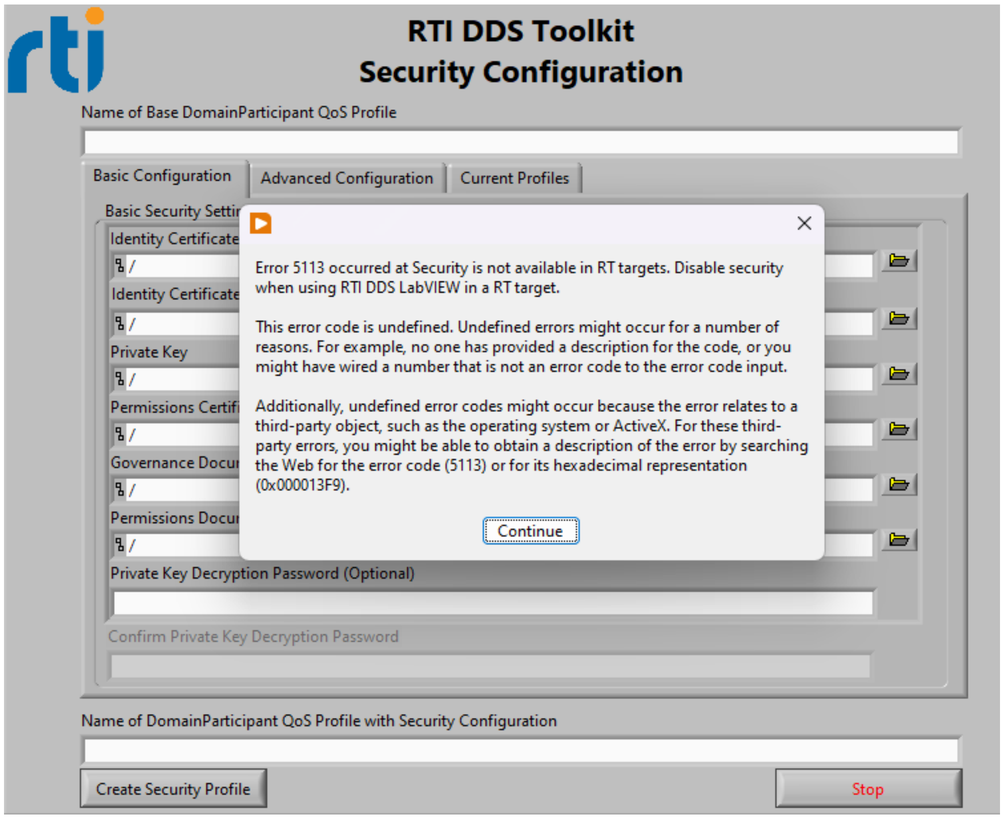
Error uninstalling the RTI DDS Toolkit RT bundle. You may need to run LabVIEW or VIPM with Administration Privileges.

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.

- Security is not supported on NI Linux RT targets. Error 5113 will be triggered if trying to enable security on an NI Linux RT target.



Error 5113 occurred at Security is not available in RT targets. Disable security when using RTI DDS LabVIEW in a RT target.

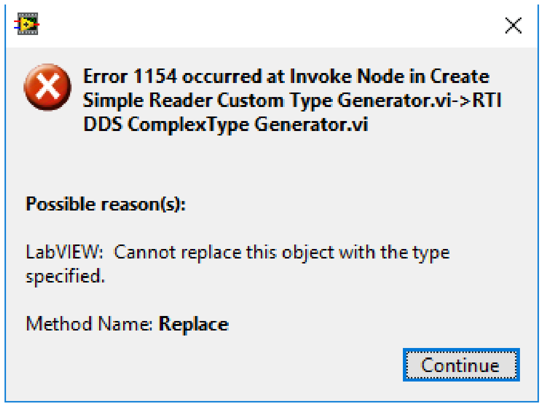
This error code is undefined. Undefined errors might occur for a number of reasons. For example, no one has provided a description for the code, or you might have wired a number that is not an error code to the error code input.

Additionally, undefined error codes might occur because the error relates to a third-party object, such as the operating system or ActiveX. For these third-party errors, you might be able to obtain a description of the error by searching the Web for the error code (5113) or for its hexadecimal representation (0x000013F9).

E.6 Errors Generating ComplexType VIs

E.6.1 Error 1154

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

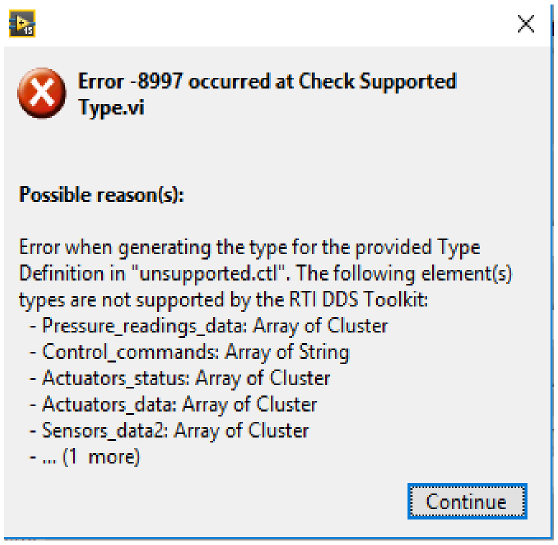
Possible reason(s):

LabVIEW: Cannot replace this object with the type specified.

Method Name: **Replace**

E.6.2 Error -8997

If you see error -8997 (shown below) when generating Complex Type VIs, the Complex Type Generator has found unsupported types inside the Type Definition (*.ctl). See Appendix C Supported Data Types and Corresponding IDL on page 147 for further information:



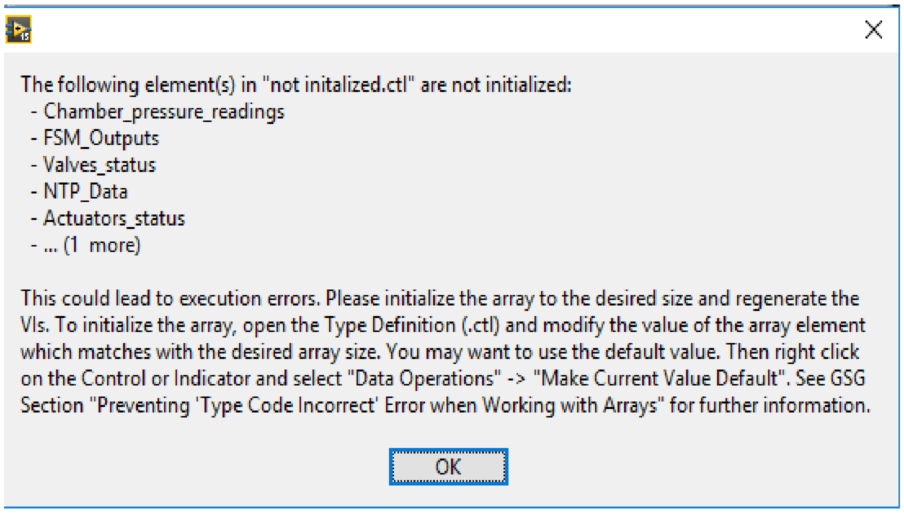
Error -8997 occurred at Check Supported Type.vi

Possible reason(s):

Error when generating the type for the provided Type Definition in "unsupported.ctl". The following element(s) types are not supported by the RTI DDS Toolkit:
- Pressure_readings_data: Array of Cluster
- Control_commands: Array of String
- Actuators_status: Array of Cluster
- Actuators_data: Array of Cluster
- Sensors_data2: Array of Cluster
- ... (1 more)

E.6.3 Unitialized Array Warning

The following warning message means that the provided Type Definition contains uninitialized arrays. To fix this issue, open the Type Definition (*.ctl) and modify the value of the array element as described in the warning message box (shown below). See 3.4.2 Preventing 'Type Code Incorrect' Error when Working with Arrays on page 27 for more information.



The following element(s) in "not initalized.ctl" are not initialized:
- Chamber_pressure_readings
- FSM_Outputs
- Valves_status
- NTP_Data
- Actuators_status
- ... (1 more)

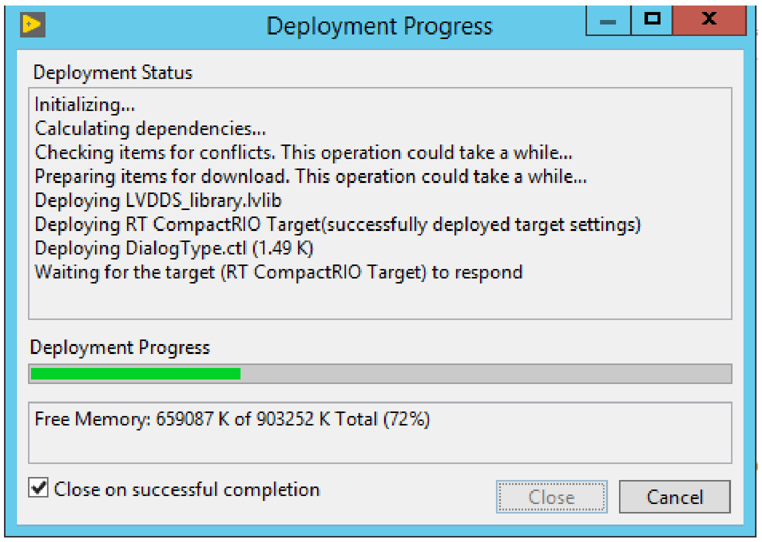
This could lead to execution errors. Please initialize the array to the desired size and regenerate the VIs. To initialize the array, open the Type Definition (.ctl) and modify the value of the array element which matches with the desired array size. You may want to use the default value. Then right click on the Control or Indicator and select "Data Operations" -> "Make Current Value Default". See GSG Section "Preventing 'Type Code Incorrect' Error when Working with Arrays" for further information.

This warning doesn’t prevent you from generating VIs for the Type Definition. If you click OK without initializing the array first as instructed, the VIs are created anyway, but you may experience errors while executing the VIs in an application.

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



```
Initializing...
Calculating dependencies...
Checking items for conflicts. This operation could take a while...
Preparing items for download. This operation could take a while...
Deploying LVDDS_library.lvlib
Deploying RT CompactRIO Target(successfully deployed target settings)
Deploying DialogType.ctl (1.49 K)
Waiting for the target (RT CompactRIO Target) to respond
```

```
Free Memory: 659087 K of 903252 K Total (72%)
```