RTI Routing Service

Getting Started Guide

Version 5.2.0

Your systems. Working as one.

Printed in U.S.A. First printing.
June 2015.

Technical Support

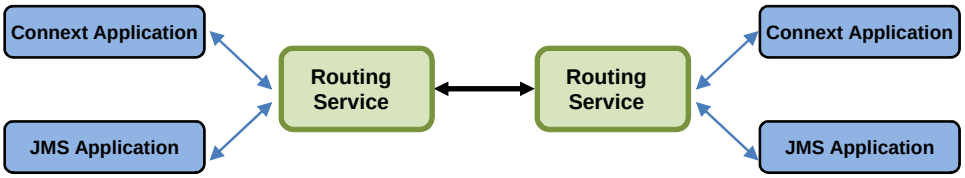
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

Contents

Chapter 1 Welcome to RTI Routing Service

Welcome to *RTI® Routing Service*, an out- of-the-box solution for integrating disparate and geographically dispersed systems. It scales *RTI Connext™ DDS* applications across domains, LANs and WANs, including firewall and NAT traversal. *Routing Service* also supports DDS-to-DDS bridging by allowing you to make transformations in the data along the way. This allows unmodified DDS applications to communicate even if they were developed using incompatible interface definitions. This is often the case when integrating new and legacy applications or independently developed systems. Using *RTI Routing Service Adapter SDK*, you can extend *Routing Service* to interface with non-DDS systems using off-the-shelf or custom developed adapters, including to third-party JMS implementations and legacy code written to the network socket API.

Traditionally, *Connext DDS* applications can only communicate with applications in the same domain. With *Routing Service*, you can send and receive data across domains. You can even transform and filter the data along the way! Not only can you change the actual data values, you can change the data's type. So the sending and receiving applications don't even need to use the same data structure. You can also control which data is sent by using allow and deny lists.



Simply set up *Routing Service* to pass data from one domain to another and specify any desired data filtering and transformations. No changes are required in the *Connext DDS* applications.

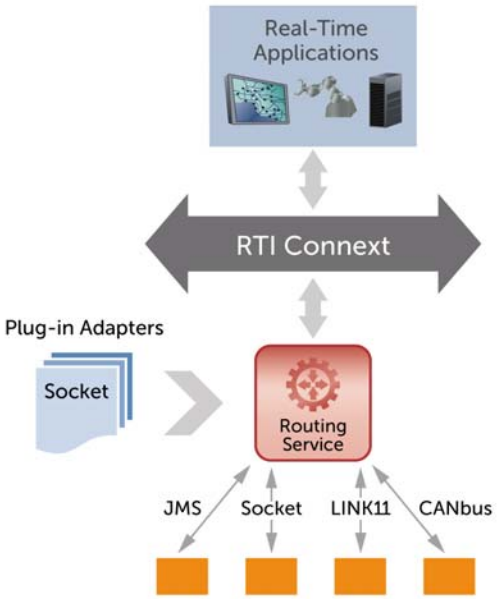
Key benefits of *Routing Service*:

- It can significantly reduce the time and effort spent integrating and scaling *Connext DDS* applications across Wide Area Networks and Systems-of-Systems.

  Many systems today already rely on *Connext DDS* to distribute their information across a Local Area Network (LAN). However, more and more of these systems are being integrated in Wide Area Networks (WANs). With *Routing Service*, you can scale *Connext DDS* real-time publish/subscribe data-distribution beyond the current local networks and make it available throughout a WAN—without making any changes to existing *Connext DDS* applications. You can take an existing, even deployed system and integrate it with new applications or other existing systems without changing those existing systems.

- With *Routing Service*, you can build modular systems out of existing systems. Data can be contained in private domains within subsystems and you can designate that only certain "global topics" can be seen across domains. The same mechanism controls the scope of discovery. Both application-level and discovery traffic can be scoped, facilitating scalable designs.
- *Routing Service* provides secure deployment across multiple sites. You can partition networks and protect them with firewalls and NATS and precisely control the flow of data between the network segments.
- It allows you to manage the evolution of your data model at the subsystem level. You can use *Routing Service* to transform data on the fly, changing topic names, type definitions, QoS, etc., seamlessly bridging different generations of topic definitions.
- *Routing Service* provides features for development, integration and testing. Multiple sites can each locally test and integrate their core application, expose selected topics of data, and accept data from remote sites to test integration connectivity, topic compatibility and specific use-cases.
- It connects remotely to live, deployed systems so you can perform live data analytics, fault condition analysis, and data verification.
- *RTI Routing Service Adapter SDK* allows you to quickly build and deploy bridges to integrate DDS and non-DDS systems. This can be done in a fraction of the time required to develop completely custom solutions. Bridges automatically inherit advanced DDS capabilities, including automatic discovery of applications; data transformation and filtering; data lifecycle management and support across operating systems; programming languages and network transports.

  RTI Routing Service Adapter SDK offers an out-of-the-box solution for interfacing with third-party protocols and technology. It includes prebuilt adapters that can be used out-of-the-box to interface with third-party Java Message Service (JMS) providers or legacy code written to the network socket API. Adapters include source code so they can be easily modified to meet application-specific requirements or serve as a template for quick creation of new custom adapters.



Quickly build and deploy bridges between natively incompatible protocols and technologies using Connext DDS

1.1 Available Documentation

Routing Service documentation includes:

- **Getting Started Guide** (RTI_Routing_Service_GettingStarted.pdf)—Highlights the benefits of *Routing Service*. It provides installation and startup instructions, and walks you through several examples so you can quickly see the benefits of using *Routing Service*.
- **Release Notes** (RTI_Routing_Service_ReleaseNotes.pdf)—Describes system requirements and compatibility, as well as any version-specific changes and known issues.
- **User's Manual** (RTI_Routing_Service_UsersManual.pdf)—Describes how to configure *Routing Service* and use it remotely.

1.2 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

  This refers to the installation directory for *Connext DDS*.

  The default installation paths are:

  - Mac OS X systems:

    /Applications/rti_connext_dds-*version*

  - UNIX-based systems, non-*root* user:

    /home/*your user name*/rti_connext_dds-*version*

  - UNIX-based systems, *root* user:

    /opt/rti_connext_dds-*version*

  - Windows systems, user without Administrator privileges:

    <*your home directory*>\rti_connext_dds-*version*

  - Windows systems, user with Administrator privileges:

    C:\Program Files\rti_connext_dds-*version* (for 64-bits machines) or
    C:\Program Files (x86)\rti_connext_dds-*version* (for 32-bit machines)

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

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

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

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

  or if you have defined the NDDSHOME environment variable:

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

❑ RTI Workspace directory, **rti_workspace**

The RTI Workspace is where all configuration files for the applications and example files are located. All configuration files and examples are copied here the first time you run *RTI Launcher* or any script in **<NDDSHOME>/bin**. The default path to the RTI Workspace directory is:

- Mac OS X systems:

  /Users/*your user name*/rti_workspace

- UNIX-based systems:

  /home/*your user name*/rti_workspace

- Windows systems:

  ***your Windows documents folder*\rti_workspace**

  Note: '*your Windows documents folder*' depends on your version of Windows. For example, on Windows 7, the folder is **C:\Users\your user name\Documents**; on Windows Server 2003, the folder is **C:\Documents and Settings\your user name\Documents**.

You can specify a different location for the **rti_workspace** directory. See the *RTI Connext DDS Core Libraries Getting Started Guide* for instructions.

❑ **<path to examples>**

Examples are copied into your home directory the first time you run *RTI Launcher* or any script in **<NDDSHOME>/bin**. This document refers to the location of these examples as **<path to examples>.** Wherever you see <path to examples>, replace it with the appropriate path.

By default, the examples are copied to **rti_workspace/*version*/examples**

So the paths are:

- Mac OS X systems:

  /Users/*your user name*/rti_workspace/*version*/examples

- UNIX-based systems:

  /home/*your user name*/rti_workspace/*version*/examples

- Windows systems:

  ***your Windows documents folder*\rti_workspace*version*\examples**

  Note: '*your Windows documents folder*' is described above.

You can specify that you do not want the examples copied to the workspace. See the *RTI Connext DDS Core Libraries Getting Started Guide* for instructions.

Chapter 2 Running Routing Service

Routing Service is installed by the *Connext DDS* package installer.

2.1 Starting Routing Service

Routing Service runs as a separate application. The script to run the executable is in **<NDDSHOME>/bin**.[1]

Routing Service supports loading Java adapters. If your configuration is set up to load a Java adapter, follow these steps:

1. **On Windows Systems:** To use a Java adapter, you must have the appropriate Visual Studio redistributable libraries installed on the target system. You can obtain this package from Microsoft or RTI (see the *RTI Connext DDS Core Libraries Release Notes* for details).
2. Make sure Java 1.7 or higher is available.
3. Make sure you add the directory of the Java Virtual Machine dynamic library to your environment variable: LD_LIBRARY_PATH (on UNIX-based systems) or Path (on Windows systems). For example:

```
setenv LD_LIBRARY_PATH
     ${LD_LIBRARY_PATH}:/local/java/jdk1.7.0/jre/lib/i386/client
```

To start Routing Service, enter:

```
<NDDSHOME>/bin/rtiroutingservice [options]
```

For example (note: you would enter this all on one line):

```
<NDDSHOME>/bin/rtiroutingservice
    -cfgFile <path to examples[1]>/routing_service/shapes/topic_bridge.xml \
    -cfgName example
```

Table 2.1 describes the command-line options.

1. See Paths Mentioned in Documentation (Section 1.2)

2.2 Stopping Routing Service

To stop *Routing Service*, press **Ctrl-c**. *Routing Service* will perform a clean shutdown.

Table 2.1 **RTI Routing Service Command-line Options**

Option	Description
-appName <name>	Assigns a name to the execution of the *Routing Service*. Remote commands and status information will refer to the routing service using this name. See the *Routing Service User's Manual* for more information. In addition, the name of *DomainParticipants* created by *Routing Service* will be based on this name. Default: The name given with **-cfgName**, if present, otherwise it is "RTI_Routing_Service".
-cfgFile <name>	Specifies a configuration file to be loaded. See How to Load the XML Configuration (Section 2.2).
-cfgName <name>	Specifies a configuration name. *Routing Service* will look for a matching <routing_service> tag in the configuration file. This parameter is required unless you use -remoteAdministrationDomainId and -noAutoEnable.
-domainIdBase <ID>	Sets the base domain ID. This value is added to the domain IDs in the configuration file. For example, if you set **-domainIdBase** to 50 and use domain IDs 0 and 1 in the configuration file, then the *Routing Service* will use domains 50 and 51. Note: -domainIdBase only affects the domain IDs of DomainRoute participants; it does not affect the domain IDs of participants used for monitoring or administration. Default: 0
-help	Displays help information.
-identifyExecution	Appends the host name and process ID to the service name provided with the **-appName** option. This helps ensure unique names for remote administration and monitoring. For example: MyRoutingService_myhost_20024
-licenseFile <file>	Specifies the license file (path and filename). Only applicable to licensed versions of *Routing Service*. If not specified, *Routing Service* looks for the license as described in Installing the License File (Section 2.3) in the Getting Started Guide.
-maxObjectsPerThread <int>	Parameter for the DomainParticipantFactory.
-noAutoEnable	Starts *Routing Service* in a disabled state. Use this option if you plan to enable *Routing Service* remotely, as described in the *User's Manual*. This option overwrites the value of the **enable** attribute in the **<routing_service>** tag.

Table 2.1 **RTI Routing Service Command-line Options**

Option	Description
-remoteAdministrationDomainId <ID>	Enables remote administration and sets the domain ID for remote communication. When remote administration is enabled, *Routing Service* will create a *DomainParticipant*, *Publisher*, *Subscriber*, *DataWriter*, and *DataReader* in the designated domain. The QoS values for these entities are described in the *Routing Service User's Manual*. This option overwrites the value of the tag **<domain_id>** within a **<administration>** tag. (See the *Routing Service User's Manual* for information on configuring remote access). Default: Remote administration is not enabled unless it is enabled from the XML file.
-remoteMonitoringDomainId <ID>	Enables remote monitoring and sets the domain ID for status publication. When remote monitoring is enabled, *Routing Service* will create one DomainParticipant, one Publisher, five DataWriters for data publication (one for each kind of entity), and five DataWriters for status publication (one for each kind of entity). The QoS values for these entities are described in the *Routing Service User's Manual.* This option overwrites the value of the tag **<domain_id>** within a **<monitoring>** tag. (See the *Routing Service User's Manual* for information on configuring remote monitoring). Default: Remote monitoring is not enabled unless it is enabled from the XML file.
-stopAfter <sec>	Stops the service after the specified number of seconds.
-use42eAlignment	Enables compatibility with *RTI Data Distribution Service* 4.2e. This option should be used when compatibility with 4.2e is required and the topic data types contain double, long long, unsigned long long, or long double members. Default: Disabled
-verbosity <n>	Controls what type of messages are logged: 0 - Silent 1 - Exceptions (*Connext DDS* and *Routing Service*) (default) 2 - Warnings (*Routing Service*) 3 - Information (*Routing Service*) 4 - Warnings (*Connext DDS* and *Routing Service*) 5 - Tracing (*Routing Service*) 6 - Tracing (*Connext DDS* and *Routing Service*) Each verbosity level, *n*, includes all the verbosity levels smaller than *n*.
-version	Prints the *Routing Service* version number.

2.3 Linking the Routing Service Library into Your Application

Routing Service can be deployed as a C library linked into your application on select architectures (see the *Release Notes*). This allows you to create, configure and start *Routing Service* instances from your application. The following code shows the typical use of the API:

```
struct RTI_RoutingServiceProperty property =
                                        RTI_RoutingServiceProperty_INITIALIZER;
struct RTI_RoutingService * service = NULL;

property.cfg_file     = "my_routing_service_cfg.xml";
property.service_name = "my_routing_service";
...
service = RTI_RoutingService_new(&property);
if(service == NULL) {
   printf("Error...");
   return -1;
}

if(!RTI_RoutingService_start(service)) {
   printf("Error...");
   RTI_RoutingService_delete(service);
   return -1;
}
while(keep_running) {
    sleep();
    ...
}
RTI_RoutingService_delete(service);
return 0;
```

To build your application, link it with the *Routing Service* library in **<NDDSHOME>/bin/<*architecture*>/**.[1] Replace <*architecture*> with an architecture string from the *Release Notes*. (Note: This process cannot be used on all architectures; see the *Release Notes* for details.)

If you are using the C API, see the example in **<path to examples[1]>/routing_service/routing_service_lib**.

Example makefiles and project files for several architectures are provided.

Also see the **README.txt** file in the **routing_service_lib/src** directory.

1. See Paths Mentioned in Documentation (Section 1.2)

Chapter 3 Using the Examples

This chapter describes several examples, all of which use *RTI Shapes Demo* to publish and subscribe to topics which are colored moving shapes (squares, circles, triangles):

- Example 1 - Routing All Data from One Domain to Another (Section 3.1)
- Example 2 - Changing Data to a Different Topic of Same Type (Section 3.2)
- Example 3 - Changing Some Values in Data (Section 3.3)
- Example 4 - Transforming the Data's Type and Topic with an Assignment Transformation (Section 3.4)
- Example 5 - Transforming Data with a Custom Transformation (Section 3.5)
- Example 6 - Using Remote Administration (Section 3.6)
- Example 7 - Monitoring (Section 3.7)
- Example 8 - Using the TCP Transport with Routing Service (Section 3.8)
- Example 9 - Using a File Adapter (Section 3.9)
- Example 10 - Bridging JMS and Connext DDS (Section 3.10)
- Example 11 - Using a Socket Adapter (Section 3.11)

In each example, you can start all the applications on the same computer or on different computers in your network.

If you don't have *Shapes Demo* installed already, you should download and install it from RTI's Downloads page (www.rti.com/downloads) or the RTI Support Portal, accessible from https://support.rti.com/ (the latter requires an account name and password). If you are not already familiar with how to start *Shapes Demo* and change its domain ID, please see the *Shapes Demo User's Manual* for details.

Important Notes:

- Please review Paths Mentioned in Documentation (Section 1.2) to understand where to find the examples (referred to as <path to examples>).
- The following instructions include commands that you will enter in a command shell. These instructions use forward slashes in directory paths, such as **bin/rtiroutingservice**. If you are using a Windows platform, replace all forward slashes in such paths with backwards slashes, such as **bin\rtiroutingservice**.
- If you run *Shapes Demo* and *Routing Service* on different machines and these machines do not communicate over multicast, you will have to set the environment variable NDDS_DISCOVERY_PEERS to enable communication. For example, assume that you run *Routing Service* on Host 1 and *Shapes Demo* on Host 2 and Host 3. In this case, the environment variable would be set as follows:

Host 1: `set NDDS_DISCOVERY_PEERS= <host2>, <host3>` (on Windows systems)

`setenv NDDS_DISCOVERY_PEERS <host2>, <host3>` (on UNIX-based systems)

Host 2: `set NDDS_DISCOVERY_PEERS=<host1>`

Host 3: `set NDDS_DISCOVERY_PEERS=<host1>`

3.1 Example 1 - Routing All Data from One Domain to Another

This example uses the default configuration file[1] for *Routing Service*, which routes all data published on domain 0 to subscribers on domain 1.

1. Start *Shapes Demo*. We'll call this the Publishing Demo. It uses domain ID 0.
2. Start a second copy of *Shapes Demo*. We'll call this the Subscribing Demo. Then:
   a. Open its Configuration dialog (under Controls).
   b. Press **Stop**.
   c. Change the domain ID to 1.
   d. Press **Start**.
3. In the Publishing Demo, publish some Squares, Circles, and Triangles.
4. In the Subscribing Demo, subscribe to Squares, Circles and Triangles.

   Notice that the Subscribing Demo does *not* receive any shapes. Since we haven't started *Routing Service* yet, data from domain 0 isn't routed to domain 1.
5. Start *Routing Service* by entering the following in a command shell:

   ```
   cd <NDDSHOME>
   bin/rtiroutingservice -cfgName default
   ```

   Now you should see all the shapes in the Subscribing Demo.
6. Stop *Routing Service* by pressing **Ctrl-c**.

   You should see that the Subscribing Demo stops receiving shapes.

Additionally, you can start *Routing Service* (Step 5) with the following parameters:

- **-verbosity 3**, to see messages from *Routing Service*, including events that have triggered the creation of routes.
- **-domainIdBase *X***, to use domains ***X*** and ***X+1*** instead of 0 and 1 (in this case, you need to change the domain IDs used by *Shapes Demo* accordingly). This option adds ***X*** to the domain IDs in the configuration file. (Note: **-domainIdBase** only affects the domain IDs of DomainRoute participants; it does not affect the domain IDs of participants used for monitoring or administration.)

1. <NDDSHOME>/resource/xml/RTI_ROUTING_SERVICE.xml

3.2 Example 2 - Changing Data to a Different Topic of Same Type

In this example, the routing service receives samples of topic Square and republishes them as samples of topic Circle.

1. Start *Shapes Demo*. We'll call this the Publishing Demo. It uses domain ID 0.
2. Start a second copy of *Shapes Demo*. We'll call this the Subscribing Demo. Then:
   a. Open its Configuration dialog (under Controls).
   b. Press ***Stop***.
   c. Change the domain ID to 1.
   d. Press **Start**.
3. Start *Routing Service* by entering the following in a command shell:

```
cd <NDDSHOME>
bin/rtiroutingservice
-cfgFile <path to examples>/routing_service/shapes/topic_bridge.xml \
-cfgName example
```

4. In the Publishing Demo (domain 0), publish some Squares, Circles and Triangles.
5. In the Subscribing Demo (domain 1), subscribe to Squares, Circles and Triangles.

   You will see that all the squares (and only squares) from domain 0 are republished as circles on domain 1.
6. Stop *Routing Service* by pressing **Ctrl-c**.
7. To see how this example is configured, review the contents of **<path to examples>/routing_service/shapes/topic_bridge.xml**.
8. Try writing your own topic route that republishes triangles on domain 0 to circles on domain 1. Create some Triangle publishers and a Circle subscriber on the respective *Shapes Demo* windows.

3.3 Example 3 - Changing Some Values in Data

So far, we have learned how to route samples from one topic to another topic of the same data type. Now we will see how to change the value of some fields in the samples and republish them.

1. Start *Shapes Demo*. We'll call this the Publishing Demo. It uses domain ID 0.
2. Start a second copy of *Shapes Demo*. We'll call this the Subscribing Demo. Then:
   a. Open its Configuration dialog (under Controls).
   b. Press **Stop**.
   c. Change the domain ID to 1.
   d. Press **Start**.

3. Start *Routing Service* by entering the following in a command shell:

```
cd <NDDSHOME>
bin/rtiroutingservice -cfgFile \
 <path to examples>/routing_service/shapes/topic_bridge_w_transf1.xml \
 -cfgName example
```

4. In the Publishing Demo (domain 0), publish some Squares.
5. In the Subscribing Demo (domain 1), subscribe to Squares.

   In the Subscribing Demo, notice that the (x,y) coordinates of the shapes are inverted from what appears in the Publishing Demo.
6. Stop *Routing Service* by pressing **Ctrl-c**.
7. To see how this example is configured, review the contents of **<path to examples>/routing_service/shapes/topic_bridge_w_transf1.xml**.
8. Try changing the transformation to assign the output **shapesize** to the input **x**.

3.4 Example 4 - Transforming the Data's Type and Topic with an Assignment Transformation

This example shows how to transform the data topic and type. We will use *rtiddsspy* to verify the result. *rtiddsspy* is a utility provided with *Connext DDS*; it monitors publications on any DDS domain.

1. Start *Shapes Demo*. We'll call this the Publishing Demo. It uses domain ID 0.
2. Start *Routing Service* by entering the following in a command shell:

```
cd <NDDSHOME>
bin/rtiroutingservice -cfgFile \
 <path to examples>/routing_service/shapes/topic_bridge_w_transf2.xml \
 -cfgName example
```

3. In the Publishing Demo (domain 0), publish some Squares.
4. We will use the *rtiddsspy* utility to verify the transformation of the data topic and type. If you have *Connext DDS* installed, run these commands:

```
cd <NDDSHOME>
bin/rtiddsspy -domainId 0 -printSample
bin/rtiddsspy -domainId 1 -printSample
```

   You will notice that the publishing samples received by *rtiddsspy* for domain 0 is of type ShapeType and topic Square. The subscribing samples received by *rtiddsspy* for domain 1 are of type Point and topic Position. Notice that the two data structures are different.
5. Stop *Routing Service* by pressing **Ctrl-c**.
6. To see how this example is configured, review the contents of **<path to examples>/routing_service/shapes/topic_bridge_w_transf2.xml**.

3.5 Example 5 - Transforming Data with a Custom Transformation

The previous example used a built-in transformation. Now we will use our own transformation between shapes. *Routing Service* allows loading shared libraries that implement the transformation API to create custom transformations. To build a custom transformation, you must have the *Connext DDS* middleware libraries installed.

1. Compile the transformation in **<path to examples>\routing_service\shapes\transformation\[make** or **windows]**:
   - On UNIX-based systems:
     - Set the environment variable NDDSHOME as described in the *RTI Connext DDS Core Libraries Getting Started Guide*.
     - Enter:

       ```
       cd <path to examples>/routing_service/shapes/transformation/make
       ```
     - Enter:

       ```
       gmake -f Makefile.<architecture>
       ```
   - On Windows systems:
     - Set the environment variable NDDSHOME as described in the *RTI Connext DDS Core Libraries Getting Started Guide.*
     - Open the Visual Studio solution under **<path to examples>\routing_service\shapes\transformation\windows**. For example, if you are using Visual Studio 2013, open **shapestransf-vs2013.sln**.
     - Select the **Release DLL** build mode.
     - Build the solution.
2. Run *Shapes Demo* and *Routing Service* as in the previous examples:
   a. Start *Shapes Demo* on domain 0 (the default domain). We'll call this the Publishing Demo.
   b. Start a second copy of *Shapes Demo*. We'll call this the Subscribing Demo. Then:
      - Open its Configuration dialog (under Controls).
      - Press **Stop**.
      - Change the domain ID to 1.
      - Press **Start**.
   c. Start *Routing Service* by entering the following in a command shell:

      ```
      cd <NDDSHOME>
      bin/rtiroutingservice -cfgFile \
<path to examples>/routing_service/shapes/topic_bridge_w_custom_transf.xml \
       -cfgName example
      ```
3. In the Publishing Demo (domain 0), publish some Squares.
4. In the Subscribing Demo (domain 1), subscribe to Squares.

   Notice that squares on domain 1 have only two possible values for **x**.
5. Stop *Routing Service* by pressing **Ctrl-c**.

6. To see how this example is configured, review the contents of **<path to examples>/routing_service/shapes/topic_bridge_w_custom_transf.xml**. Notice how the transformation is instantiated inside the topic route.
7. Change the fixed 'x' values for the Squares in the configuration file and restart *Routing Service*.
8. Stop *Routing Service* by pressing **Ctrl-c**.
9. Edit the source code to make the transformation multiply the value of the field by the given integer constant instead of assigning the constant.

   Hint: Look for the function **ShapesTransformationPlugin_createOutputSample()**, called from **ShapesTransformation_transform()** and use **DDS_DynamicData_get_long()** before **DDS_DynamicData_set_long().**
10. Recompile the transformation (the new shared library will be copied automatically) and run *Routing Service* as before.

3.6 Example 6 - Using Remote Administration

In this example, we will configure *Routing Service* remotely. We won't see data being routed until we remotely enable an auto topic route after the application is started. Then we will change a QoS value and see that it takes effect on the fly.

1. Start *Shapes Demo*. We'll call this the Publishing Demo. It uses domain ID 0.
2. Start a second copy of *Shapes Demo*. We'll call this the Subscribing Demo. Then:
   a. Open its Configuration dialog (under Controls).
   b. Press **Stop**.
   c. Change the domain ID to 1.
   d. Press **Start**.
3. Start *Routing Service* by entering the following in a command shell:

```
cd <NDDSHOME>
bin/rtiroutingservice \
 -cfgFile <path to examples>/routing_service/shapes/administration.xml \
 -appName MyRoutingService -cfgName example
```

4. In the Publishing Demo (domain 0), publish some Squares, Circles and Triangles.
5. In the Subscribing Demo (domain 1), subscribe to Squares, Circles and Triangles.

   Notice that no data is routed to domain 1.
6. On a different or the same machine, start the *Routing Service* shell:

```
cd <NDDSHOME>
bin/rtirssh -domainId 0
```

   Note: We use domain 0 in the shell because *Routing Service* is configured in **administration.xml** to receive remote commands on that domain. You could have started *Routing Service* with the **-remoteAdministrationDomainId X** command-line option and then used domain **X** for the shell.

7. In the shell, enter the following command:

```
> enable MyRoutingService RemoteConfigExample::Session::Shapes
```

   Notice that the shapes are now received on domain 1. The above command consists of two parts: the name of the routing service, which you gave when you launched the application with the option **-appName**, and the name of the entity you wanted to enable. That name is formed by appending its parent entities' names starting from the domain route as defined in the configuration file **administration.xml**.

   You could have run *Routing Service* without **-appName**. The name would have been the one provided with **-cfgName** ("example"). You could also have used **-identifyExecution** to generate the name based on the host and application ID. In this case, you would have used this automatic name in the shell.

8. Examine the file **<path to examples>/routing_service/shapes/time_filter_qos.xml** on the routing service machine. It contains an XML snippet that defines a QoS value for an auto topic route's DataReader. Execute the following command in the shell:

```
> update MyRoutingService RemoteConfigExample::Session::Shapes \
  <path to examples>/routing_service/shapes/time_filter_qos.xml
```

   Notice that the receiving application only gets shapes every 2 seconds. The auto topic route has been configured to read (and forward) samples with a minimum separation of 2 seconds.

   Routing Service can be configured remotely using files located on the routing service machine or the shell machine. In step 9 you will edit the configuration files on both machines. Step 10 shows how to specify which of the two configuration files you want to use. ***If you are running the shell and Routing Service on the same machine, skip steps 9 and 10.***

9. Edit the XML configuration files on both machines:

   a. In **<path to examples>/routing_service/shapes/time_filter_qos.xml** on the routing service machine, change the minimum separation to 0 seconds.

   b. In **<path to examples>/routing_service/shapes/time_filter_qos.xml** on the shell machine, change the minimum separation to 5 seconds.

10. Run the following commands in the shell:

   a. Enter the following command. Notice the use of **remote** at the end—this means you want to use the XML file on the routing service machine (the *remote* machine, which is the default if nothing specified).

```
> update MyRoutingService RemoteConfigExample::Session::Shapes \
<path to examples>/routing_service/shapes/time_filter_qos.xml remote
```

   Note: The path to the XML file in this example is relative to the working directory from which you run *Routing Service*.

   Since no time filter applies, the shapes are received as they are published.

   b. Enter the following command. This time we use **local** at the end—this means you want to use the XML file on the shell machine (the *local* machine).

```
> update MyRoutingService RemoteConfigExample::Session::Shapes \
  <path to examples>/routing_service/shapes/time_filter_qos.xml local
```

   Note: The path to the XML file in this example is relative to the working directory from which you run the *Routing Service* shell.

You will see that now the shapes are only received every 5 seconds.

c. Enter the following command. Once again, we use **remote** at the end to switch back to the XML file on the routing service machine.

```
> update MyRoutingService RemoteConfigExample::Session::Shapes \
<path to examples>/routing_service/shapes/time_filter_qos.xml remote
```

Shapes are once again received as they are published.

11. Disable the auto topic route again by entering:

```
> disable MyRoutingService RemoteConfigExample::Session::Shapes
```

The shapes are no longer received on Domain 1.

Note: At this point, you could still update the auto topic route's configuration. You could also change immutable QoS values, since the DataWriter and DataReader haven't been created yet. These changes would take effect the next time you called enable.

12. Run these commands in the shell and see what happens after each one:

```
> enable MyRoutingService RemoteConfigExample::Session::SquaresToCircles
> disable MyRoutingService RemoteConfigExample::Session::SquaresToCircles
> enable MyRoutingService RemoteConfigExample::Session::SquaresToTriangles
```

These commands change the output topic that is published after receiving the input Square topic. As you can see, you can use the shell to switch topic routes after *Routing Service* has been started.

13. Stop the shell by running this command in the shell:

```
> exit
```

14. Stop *Routing Service* by pressing **Ctrl-c**.

3.7 Example 7 - Monitoring

With *Routing Service* you can publish status information. The monitoring configuration is quite flexible and allows selecting the entities that you want to monitor and how often they should publish their status.

1. Start *Shapes Demo*. We'll call this the Subscribing Demo. Then:

a. Open its Configuration dialog (under Controls).

b. Press **Stop**.

c. Change the domain ID to 1.

d. Press **Start**.

2. Start a second copy of *Shapes Demo*. We'll call this the Publishing Demo. Then:

a. Open its Configuration dialog (under Controls).

b. Press **Stop**.

c. Change the domain ID to 0.

d. Press **Start**.

3. In the Publishing Demo (domain 0), publish two Squares, two Circles, and two Triangles.

4. In the Subscribing Demo (domain 1), subscribe to Squares, Circles and Triangles. At this point you will not see any shapes moving in the Subscribing Demo. It isn't receiving shapes from the Publishing Demo because they use different domain IDs.

5. Start *Routing Service* by entering the following in a command shell:

```
cd <NDDSHOME>
bin/rtiroutingservice \
    -cfgFile <path to examples>/routing_service/shapes/monitoring.xml \
    -cfgName example -appName MyRoutingService
```

   This configuration file routes Squares and Circles using two different topic routes.

6. Now you can subscribe to the monitoring topics (see Chapter 5 in the *Routing Service User's Manual* for more information). You can do it in your own application, or by using *RTI Spreadsheet Add-in for Microsoft® Excel®* or *rtiddsspy*. We will use *rtiddsspy*, a utility provided with *Connext DDS* that monitors publications on any DDS domain. If you have *Connext DDS* installed, enter the following in a command shell:

```
cd <NDDSHOME>
bin/rtiddsspy -domainId 2 -printSample
```

   Note: We use domain 2 in *Connext DDS*because *Routing Service* is configured in **monitoring.xml** to publish status information on that domain. You could have started *Routing Service* with the **-remoteMonitoringDomainId X** command-line option and then used domain *X* for *rtiddsspy*.

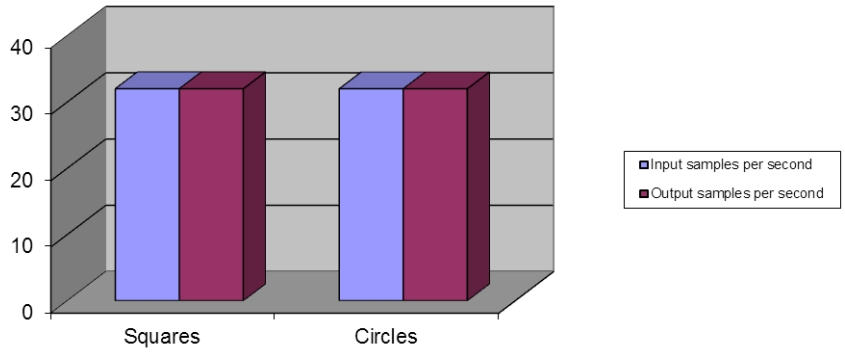
7. Depending on the publication period of the entity in the XML file we used, you will receive status samples at different rates. In the output from *rtiddsspy*, check the statistics about the two topic routes we are using. We will focus on the input samples per second:

```
routing_service_name: "MyRoutingService"
domain_route_name: "DomainRoute"
session_name: "Session"
name: "Squares"
input_samples_per_s:
   publication_period_metrics:
      period_ms: 2000
      count: 62
      mean: 30.969030
      minimum: 29.970030
      maximum: 31.968033
      std_dev: 0.999001
  historical_metrics:

routing_service_name: "MyRoutingService"
domain_route_name: "DomainRoute"
session_name: "Session"
name: "Circles"
input_samples_per_s:
   publication_period_metrics:
      period_ms: 5000
      count: 158
      mean: 31.574739
      minimum: 29.970030
      maximum: 32.000000
      std_dev: 0.802551
   historical_metrics:
```

The number of samples per second in our case is 32. That value depends on the publication rate of *Shapes Demo*, configurable with the option **-pubInterval <milliseconds between writes>**.

8. *Optional*: If you have *RTI Spreadsheet Add-in for Microsoft Excel*[1], open **<path to examples>/routing_service/shapes/monitoring_visualization.xls**. Select the **Topic Route** worksheet (from the tabs at the bottom of Excel); you should see the following bar chart, among other data and figures:



9. Create two additional Square publishers in the Publishing Demo (domain 0).

10. Check *rtiddsspy* again for new status information:

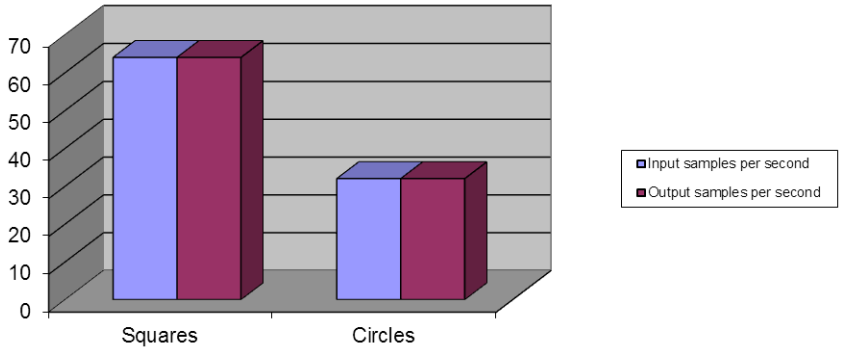
```
routing_service_name: "MyRoutingService"
domain_route_name: "DomainRoute"
session_name: "Session"
name: "Squares"
input_samples_per_s:
   publication_period_metrics:
      period_ms: 2000
      count: 128
      mean: 63.968018
      minimum: 63.936066
      maximum: 64.000000
      std_dev: 0.031968
   historical_metrics:

routing_service_name: "MyRoutingService"
domain_route_name: "DomainRoute"
session_name: "Session"
name: "Circles"
input_samples_per_s:
   publication_period_metrics:
      period_ms: 5000
      count: 160
      mean: 31.974421
      minimum: 31.968033
      maximum: 32.000000
      std_dev: 0.012783
   historical_metrics:
```

In the topic route, **Squares**, we are receiving double amount of data.

1. *RTI Spreadsheet Add-in for Microsoft Excel* is a separate tool included with *RTI Connext DDS Professional.*

11. *Optional*: If you have *Spreadsheet Add-in for Microsoft Excel* running from Step 8, notice the change in the bar graph:



12. Look at the status of the domain route in the output from *rtiddsspy*:

```
routing_service_name: "MyRoutingService"
domain_route_name: "DomainRoute"
name: "Session"
input_samples_per_s:
   publication_period_metrics:
      period_ms: 5000
      count: 480
      mean: 47.961632
      minimum: 31.968033
      maximum: 64.000000
      std_dev: 0.019175
   historical_metrics:
```

It contains an aggregation of the two contained topic routes, giving us a mean of nearly 48 samples per second.

13. We can update the monitoring configuration at run time using the remote administration feature. In the configuration file, we enabled remote administration on domain 0.

On a different or the same machine, start the *Routing Service* shell:

```
cd <NDDSHOME>
bin/rtirssh -domainId 0
```

14. We are receiving the status of the topic route Circles every five seconds. To receive it more often, use the following command:

```
> update MyRoutingService DomainRoute::Session::Circles
     topic_route.entity_monitoring.status_publication_period.sec=2
```

15. In some cases, you might want to know only about one specific topic route. If you only want to know about the topic route Circles but not Squares, you can disable monitoring for Squares:

```
> update MyRoutingService DomainRoute::Session::Squares
     topic_route.entity_monitoring.enabled=false
```

16. To enable it again, enter:

```
> update MyRoutingService DomainRoute::Session::Squares
     topic_route.entity_monitoring.enabled=true
```

17. If you are no longer interested in monitoring this routing service, you can completely disable it with the following command:

```
> update MyRoutingService routing_service.monitoring.enabled=false
```

Now you won't receive any more status samples.

18. You can enable it again any time by entering:

```
> update MyRoutingService routing_service.monitoring.enabled=true
```

19. Stop *rtiddsspy* by pressing **Ctrl-c.**

20. Stop the shell:

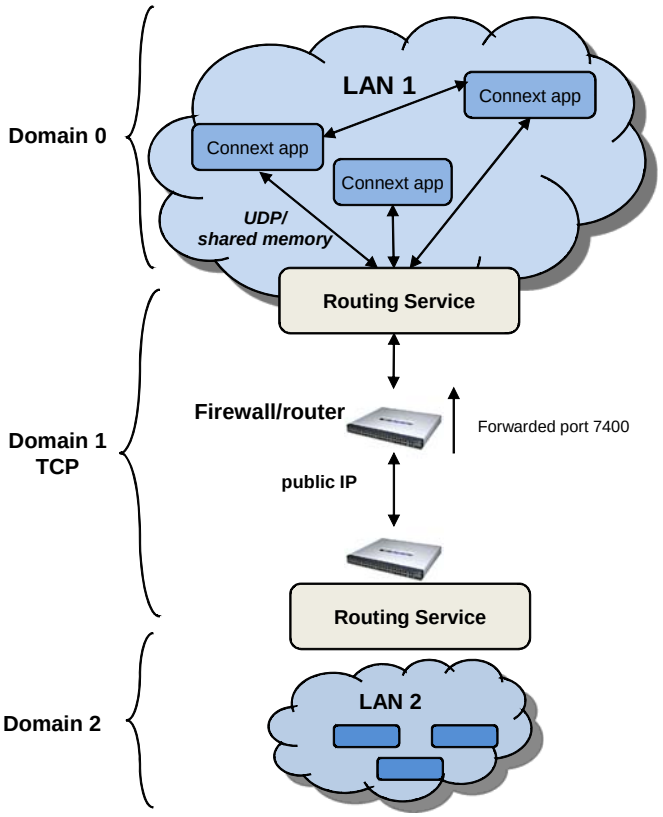
```
> exit
```

21. Stop *Routing Service* by pressing **Ctrl-c**.

3.8 Example 8 - Using the TCP Transport with Routing Service

This example shows how to use *Routing Service* to bridge data between different LANs over TCP. *Routing Service* will act as the gateway in a LAN with which other *Connext DDS* applications can communicate to send or receive data. Chapter 7 of the *Routing Service User's Manual* has more information about scenarios and detailed configuration parameters.

You will run two copies of *Routing Service*. One copy will run on a machine that is behind a firewall/router with a public IP (First Peer); the other will run on a machine in another LAN (Second Peer).



On the First Peer (behind a firewall/router with a public IP):

1. In the First Peer's network, configure the firewall to forward the TCP ports used by *Routing Service*.

   In this example, we will use port 7400.

   You do not need to configure your firewall for every single *Connext DDS* application in your LAN; doing it just once for *Routing Service* will allow other applications to communicate through the firewall.

2. Include the Second Peer's public IP address and port in the **NDDS_DISCOVERY_PEERS** environment variable.

   For example, on a UNIX-based system:

   ```
   setenv NDDS_DISCOVERY_PEERS
   tcpv4_wan://<server's public IP address>:<port>
   ```

   On a Windows system:

   ```
   set NDDS_DISCOVERY_PEERS=
   tcpv4_wan://<server's public IP address>:<port>
   ```

   When you configure **NDDS_DISCOVERY_PEERS**, make sure to use a transport class prefix (tcpv4_wan, udpv4, shmem) for each entry. (See Section 12.2 in the *RTI Connext DDS Core Libraries User's Manual* for details on formatting addresses in **NDDS_DISCOVERY_PEERS**.)

   For example:

   ```
   setenv NDDS_DISCOVERY_PEERS tcpv4_wan://10.10.1.10:7400,\
        udpv4://192.168.0.1,udpv4://192.168.0.2,shmem://
   ```

3. Set the public IP address and port in the configuration file:

   a. Open the file **<path to examples>/routing_service/shapes/tcp_transport.xml**.

   b. The file contains several routing service configurations. Find the routing service configuration, <routing_service name="TCP_1">. Then find the "public_address" property (<name>dds.transport.TCPv4.tcp1.public_address</name>) within that configuration.

   c. Set the *local* public IP address and port. For example, to set the address to 10.10.1.150 and port 7400:

   ```
    <element>
        <name>dds.transport.TCPv4.tcp1.public_address</name>
        <value>10.10.1.150:7400</value>
   </element>
   ```

   d. Save and close the file.

4. Run these commands and choose "**TCP_1**":

   ```
   cd <NDDSHOME>
   bin/rtiroutingservice \
    -cfgFile <path to examples>/routing_service/shapes/tcp_transport.xml \
    -cfgName TCP_1
   ```

5. On any computer in this LAN, start *Shapes Demo* and publish some shapes on domain 0.

On the Second Peer (a machine in any other LAN):

6. Include the First Peer's public IP address and port in the **NDDS_DISCOVERY_PEERS** environment variable the same way you did before.
7. Set the public IP address and port in the configuration file:
   a. The file contains several routing service configurations. Find the routing service configuration, **<routing_service name="TCP_2">**. Then find the "public_address" property (<name>dds.transport.TCPv4.tcp1.public_address</name>) within that configuration.
   b. Set the *local* public IP address and port. For example, to set the address to 10.10.1.10 and port 7400:

```
<element>
  <name>dds.transport.TCPv4.tcp1.public_address</name>
  <value>10.10.1.10:7400</value>
</element>
```

   c. Save and close the file.
8. Run these commands and choose "**TCP_2**":

```
cd <NDDSHOME>
bin/rtiroutingservice \
  -cfgFile <path to examples>/routing_service/shapes/tcp_transport.xml \
  -cfgName TCP_2
```

9. On any computer in this LAN, start *Shapes Demo* and create subscribers on domain 2. Do not use an already running instance of *Shapes Demo*—you need a new one that uses a different domain ID.

   You should receive what is being published in the server's LAN.

Notes:

- Running *Shapes Demo* on a Different Computer

  If the computer running *Shapes Demo* is different than the computer running the client routing service, add the address of the client (IP address or host name) to the *Shapes Demo* discovery peers before starting the shapes demo. To do so, use the **–peer** command-line option or set the **NDDS_DISCOVERY_PEERS** environment variable.

- Using Two Computers in the Same LAN

  If both machines are in the same LAN, run both routing services with the configuration file **tcp_transport_lan.xml** and use **"tcpv4_lan://"** as the peer prefix in the environment variable NDDS_DISCOVERY_PEERS. You don't need to specify an IP address in the configuration file.

- Running the Example on One Computer

  To run the example on the same machine, open the file **<path to examples>/routing_service/shapes/tcp_transport_lan.xml** and change the property **dds.transport.TCPv4.tcp1.server_bind_port** within **TCP_1** to 7401. Run both routing services with the modified **tcp_transport_lan.xml** configuration file and use **"tcpv4_lan://"** as the peer prefix in the environment variable NDDS_DISCOVERY_PEERS. You will also need to specify port 7401 in the **tcpv4_lan** peer in the NDDS_DISCOVERY_PEERS environment variable of the routing service in the Second Peer to reflect this port change in the configuration file.

❑ Using a Secure Connection over WAN

To run this example, you need OpenSSL 0.9.8n (or higher) and *RTI TLS Support*. To purchase *RTI TLS Support*, contact your account representative or **sales@rti.com**. OpenSSL is available from the RTI's Downloads page (www.rti.com/downloads), or you may obtain it from another source. Make sure the OpenSSL libraries' location is in your LD_LIBRARY_PATH (on UNIX-based systems) or Path (on Windows systems).

To run the example using a secure connection between the two router instances, use the configuration file **tcp_transport_tls.xml**. You will also need to set the peer prefix to "**tlsv4_wan://**" in the NDDS_DISCOVERY_PEERS environment variable. The **tcp_transport_tls.xml** file is based on **tcp_transport.xml** and uses a WAN configuration to establish communication.

❑ Using a Secure Connection over LAN

To run this example using a secure connection between two routers instances within the same LAN, you need OpenSSL 0.9.8n (or higher) and *TLS Support*. To purchase *TLS Support*, contact your account representative or **sales@rti.com**. OpenSSL is available from the RTI's Downloads page (www.rti.com/downloads), or you may obtain it from another source. Make sure the OpenSSL libraries' location is in your LD_LIBRARY_PATH (on UNIX-based systems) or Path (on Windows systems).

To use TLS encryption over a LAN configuration, you can use the file **tcp_transport_tls_lan.xml**. You will also need to set the peer prefix to "**tlsv4_lan://**" in the NDDS_DISCOVERY_PEERS environment variable. The **tcp_transport_tls_lan.xml** configuration file is based on **tcp_transport_lan.xml** and uses a LAN configuration to establish communication.

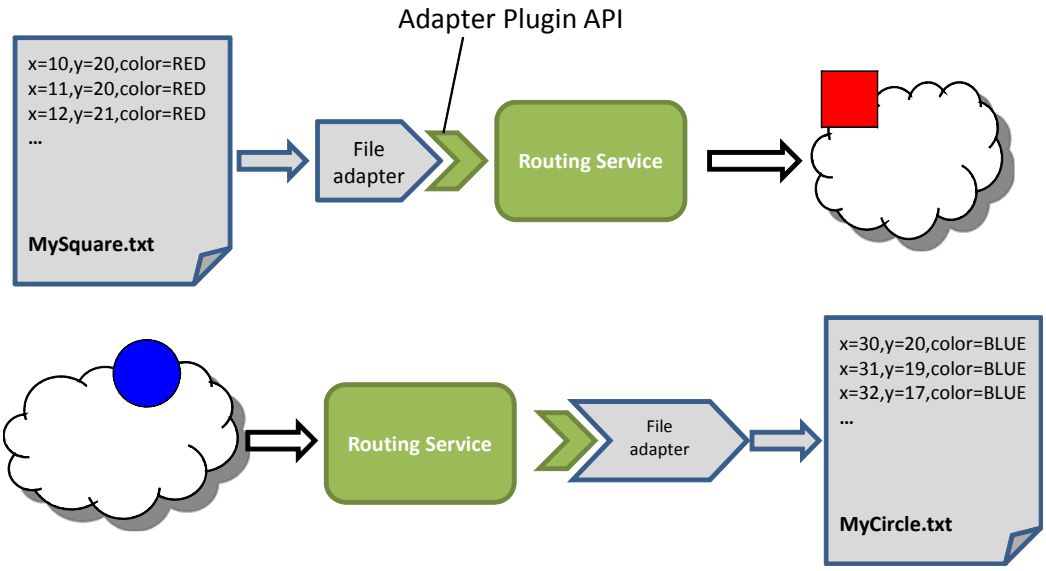
3.9 Example 9 - Using a File Adapter

The previous examples showed how to use *Routing Service*. In this one you will learn how to use an adapter, specifically *RTI Routing Service Adapter SDK*, to write and read data from files. *Routing Service* allows bridging data from different data domains with a pluggable adapter interface.

To learn how to implement your own adapter, you can follow this example and the next examples and inspect the code that is distributed with these adapters. You can also start your own adapter from scratch and follow step-by-step instructions to get a basic implementation up and running. These instructions are in the *Routing Service User's Manual* (Section 8.3).

The file adapter can read data from files with a specific format and provide it to *Routing Service*, or receive data from *Routing Service* and write it into files.

In this example, we will first write topic data (a colored square and circle) into a file and then use that file to write it back into *Connext DDS*, allowing us to modify the data with a text editor.



To run this example, you must have *Routing Service Adapter SDK* installed.

Compile the adapter

1. Compile the file adapter in **adapters/file/src:**

   #### On UNIX-based systems:

   - Set the NDDSHOME environment variable as described in the *RTI Connext DDS Core Libraries Getting Started Guide*.
   - Enter:

     ```
     cd <path to examples>/routing_service/adapters/file/make
     gmake -f Makefile.<architecture>
     ```

     The adapter shared library, **libfileadapter.so**, will be copied to **<path to examples>/routing_service/adapters/file**.

   #### On Windows systems:

   - Set the NDDSHOME environment variable as described in the *RTI Connext DDS Core Libraries Getting Started Guide*.
   - Open the Visual Studio solution under **<path to examples>/routing_service/adapters/file/windows**. For example, if you are using Visual Studio 2013, open **fileadapter-vs2013.sln**.
   - Build the solution.

   The adapter shared library, **fileadapter.dll**, will be copied to **<path to examples>/routing_service/adapters/file**.

From Connext DDS to files:

2. Run *Shapes Demo* and *Routing Service* as in the previous examples:
   a. Start *Shapes Demo* on domain 0 (the default domain).
   b. Start *Routing Service* by entering the following in a command shell:

```
cd <NDDSHOME>
bin/rtiroutingservice \
  -cfgFile <path to examples>/routing_service/shapes/file_bridge.xml \
  -cfgName dds_to_file
```

3. In *Shapes Demo*, publish some Squares.
4. Wait a few seconds and then stop *Routing Service* by pressing **Ctrl-c**.
5. A file called **MySquare.txt** should have been created in the current directory. Open it with a text editor of your choice. It should contain several lines, each consisting of a list of <field>=<value> elements. Each line represents a sample (Square) published by *Shapes Demo* and written by *Routing Service* and the file adapter.
6. On UNIX-based systems, you can see how new samples are appended to the file by running **tail -f MySquare.txt** without stopping *Routing Service*.
7. We have seen how an "output" adapter works. Open the configuration file and look for <routing_service name="dds_to_file"> to see the configuration we have just run.

From a file to Connext DDS:

8. In *Shapes Demo,* delete the Square publisher and create a Square subscriber.
9. Run *Routing Service* as in the previous examples:

```
cd <NDDSHOME>
bin/rtiroutingservice \
  -cfgFile <path to examples>/routing_service/shapes/file_bridge.xml \
  -cfgName file_to_dds
```

10. You should see squares being received by *Shapes Demo.* These samples come from what we recorded before.
11. You might have noticed than the rate at which the shape moves is much slower. This is the rate at which the file adapter is providing data to *Routing Service*. To change this rate, open **<path to examples>/routing_service/shapes/file_bridge.xml** and look for <route name="square_file"> within <routing service name="file_to_dds">. In the <property> tag change the property **ReadPeriod** from 1000 (milliseconds) to 100.
12. Stop and start again *Routing Service* as described in Step 9. The squares should be received and displayed about 10 times faster.
13. Other properties that you can configure in the file adapter are: FileName, MaxSampleSize, Loop and SamplesPerRead and, in the <output>, FileName, Flush and WriteMode.
14. You can also edit the text file and publish the new data. Open **MySquare.txt** and replace all the occurrences of "shapesize=30" with "shapesize=100".
15. Stop and start again *Routing Service* as described in Step 9. The squares will have the same position and color, but they will be bigger now.

Customize the file adapter

In the example, the file adapters use a specific format, which you already saw in the file **MySquare.txt**. Now try adapting the example to your own format.

16. The code that reads/writes from the file is in **adapters/file/src/LineConversion.c.**
17. Edit the function **RTI_RoutingServiceFileAdapter_read_sample** to implement how file data maps into a sample.
18. Edit the function **RTI_RoutingServiceFileAdapter_write_sample** to implement how a sample is written to a file.
19. Compile the code as described in Step 1 on page 3-16.

3.10 Example 10 - Bridging JMS and Connext DDS

As we saw in the previous example, *Routing Service* can load adapter plugins to read and write data from different sources. We used a file adapter implemented in C. *Routing Service* also supports Java plugins.

The adapter introduced in this chapter interfaces with JMS to produce and consume messages in *Routing Service*, allowing you to bridge JMS with other data sources, like *Connext DDS*.

This example shows how a JMS application can produce messages representing colored shapes with random positions and how *Shapes Demo* can receive and display the equivalent *Connext DDS* samples. This is possible thanks to a running instance of *Routing Service* bridging JMS and *Connext DDS* using the adapter. The opposite scenario, in which DDS samples are received by a JMS consumer, will also work.

To run this example, you must have *RTI Routing Service Adapter SDK* installed, plus *RTI Message Service* or an equivalent JMS implementation.

Compile the JMS Adapter:

To compile the JMS adapter, use the build script described below:

1. The script expects the following variables to be set. They can be set by editing the script file or environment variables.
   a. JAVA_HOME must point to a Java JDK installation (1.7 or higher)
   b. JAVAEEJAR must contain the full path and filename of the Java EE library (for example, **/myjava/j2ee/lib/javaee.jar**)
2. After setting the values for the variables, run the **build** script.

   On UNIX-based systems:

   ```
   cd <path to examples>/routing_service/adapters/jms
   ./build
   ```

   On Windows systems:

   ```
   cd <path to examples>\routing_service\adapters\jms
   build
   ```

   A Java JAR file, **jmsadapter.jar**, will be generated in the directory, **objs**.

Compile the Example JMS Publisher and Subscriber Applications:

Another build script can be used to build the JMS publisher and subscriber example.

3. The script expects the same variables as Step 1.
4. After setting the values for the variables, run the **build** script:

   On UNIX-based systems:

   ```
   cd <path to examples>/routing_service/shapes/jmsPubSub
   ./build
   ```

   On Windows systems:

   ```
   cd <path to examples>\routing_service\shapes\jmsPubSub
   build
   ```

   The class files will be generated in the directory **objs**.

Run the Example JMS and Connext DDS Publisher and Subscribers:

5. The JMS applications expect the same environment variables as in Step 1 plus the following:

   JMS_CLASSPATH must contain the classpath required to run the example with a JMS vendor of your choice. In our example, we use *RTI Messaging Service*, but you could choose a different vendor such as JBoss:

   On UNIX-based systems (enter the following all on one line):

   ```
   setenv JMS_CLASSPATH
         $NDDSHOME/lib/java/nddsjava.jar:
         $NDDSHOME/lib/java/rtijms.jar
   ```

   On Windows systems (enter the following all on one line):

   ```
   set JMS_CLASSPATH=
         %NDDSHOME%\lib\java\nddsjava.jar:
         %NDDSHOME%\lib\java\rtijms.jar
   ```

6. Run the following applications on the same or different machines:

   a. A JMS application publishing green Squares at random coordinates. Set the same environmental variables mentioned in Step 5 and include in your LD_LIBRARY_PATH (on UNIX-based platforms) or in your Path environment variable (on Windows platforms) $NDDSHOME\lib\<arch>. Then enter:

      On UNIX-based systems:

      ```
      cd <path to examples>/routing_service/shapes/jmsPubSub
      ./run Pub Shapes/Square Green 15
      ```

      On Windows systems:

      ```
      cd <path to examples>\routing_service\shapes\jmsPubSub
      run Pub Shapes/Square Green 15
      ```

   b. A JMS application subscribing to Circles. Set the same environmental variables mentioned in Step 5 and include in your LD_LIBRARY_PATH (on UNIX-based platforms) or in your Path environment variable (on Windows platforms) $NDDSHOME\lib\<arch>. Then enter:

      On UNIX-based systems:

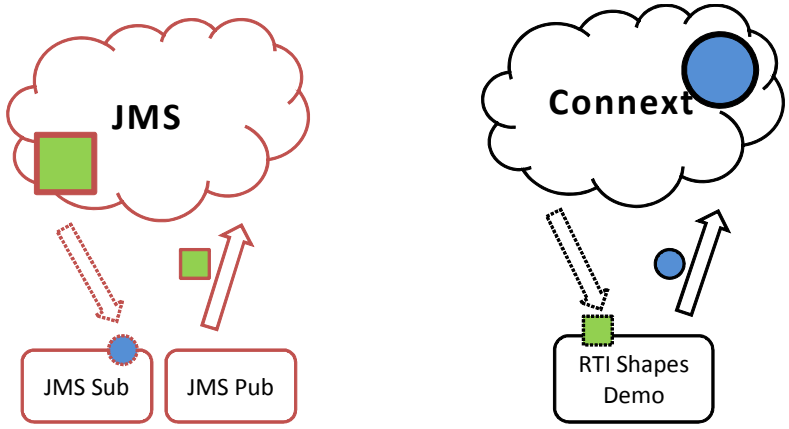
      ```
      cd <path to examples>/routing_service/shapes/jmsPubSub
      ./run Sub Shapes/Circle
      ```

      On Windows systems:

      ```
      cd <path to examples>\routing_service\shapes\jmsPubSub
      run Sub Shapes/Circle
      ```

   c. A *Connext DDS* application publishing blue Circles and subscribing to Squares. Start *Shapes Demo* on domain 0 and create a blue Circle publisher and a Square subscriber.

Now we have the following scenario, in which none of the subscribers are receiving data yet.



Configure and Run Routing Service with the JMS Adapter:

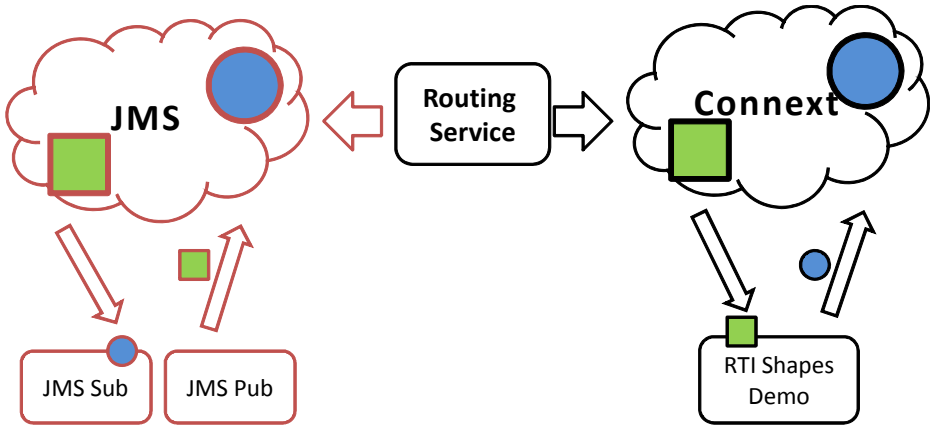
We are going to use the configuration file **<path to examples>/routing_service/shapes/jms_rti_shapes.xml** to run *Routing Service* and the JMS adapter with *RTI Message Service.*

7. Our example *Routing Service* configuration file expects the environment variables JAVAEEJAR and JMS_CLASSPATH as set in Step 5. To use different configuration or set up a different JMS vendor, you can edit the file and change the values of <jvm>/<class_path>, and the JMS connection <property> under the different <domain_route> tags.
8. When using Java adapters with *Routing Service*, the Java Virtual Machine shared library (**libjvm.so** on UNIX-based systems or **jvm.dll** on Windows Systems) must be accessible at run-time. Add its location to LD_LIBRARY_PATH on UNIX-based systems or to the Path on Windows systems. The location will depend on your architecture. It is *typically* located in:
   - On UNIX-based systems: **$JAVA_HOME/jre/lib/client**
   - On Windows systems: **%JAVA_HOME%\jre\bin\client**
9. Run the following command to start *Routing Service*[1]:

```
cd <path_to_examples>\routing_service\adapters\jms
$NDDSHOME\bin\rtiroutingservice -cfgFile \
  <path to examples>/routing_service/shapes/jms_rti.xml -cfgName jmsdds
```

1. In addition to jms_rti.xml, you can find in the same directory the file jms_openjms.xml, which includes the configuration to run the JMS adapter with OpenJMS as the JMS vendor.

10. *Routing Service* will bridge between JMS and *Connext DDS*: the JMS subscriber application will receive the blue Circles published by the *Shapes Demo* publisher, and the *Shapes Demo* subscriber will receive the green Squares published by the JMS publisher application.



How to customize the JMS adapter:

The implementation of the JMS adapter is meant to be extended to better fit your specific type system. This example will only work with JMS MapMessages and DDS types with top-level members that are of primitive or string types.

The fundamental point of extension is the translation between JMS messages and *Connext DDS* DynamicData samples. To customize this translation, follow these steps:

11. To translate from JMS messages to DynamicData samples, implement the following interface (let's call your class **mypackage.MyJMSToDynamicData)**:

```
com.rti.routingservice.adapter.jms.JMSToDynamicDataMessageTranslation
```

12. To translate from DynamicData samples to JMS messages, implement the following interface (let's call your class **mypackage.MyDynamicDataToJMS**):

```
com.rti.routingservice.adapter.jms.DynamicDataToJMSMessageTranslation
```

    You can find the default implementations in **adapters/jms/src/**.

13. Modify your routing service configuration file to tell the JMS adapter that it should instantiate and use your classes:

    a. In a <route> that reads from the JMS adapter, add a property below <input> with the name **JMSDynamicDataTranslator** and the value **mypackage.MyJMSToDynamicData**.

    b. In a <route> that writes to the JMS adapter, add a property below <output> with the name **DynamicDataJMSTranslator** and the value **mypackage.MyDynamicDataToJMS**.

    c. Make sure *mypackage* reachable in your classpath (within the <jvm>/<classpath> tag in your configuration file or in the environment variable CLASSPATH).

    See **<path to examples>/routing_service/shapes/jms_rti.xml** for more details.

Optional Exercise:

If you have completed this example and the previous one which introduced the file adapter, as an optional exercise you can make these two adapters work together. Write a routing service

configuration file that does the same thing as the example file, but have it read and write from/to JMS instead of DDS.

Hint: Your <domain_route> will contain two connections, each instantiating one of the adapter plugins (<connection_1 plugin_name="adapters::jms">, and <connection_2 plugin_name="adapters::file">).

3.11 Example 11 - Using a Socket Adapter

To run this example, you must have *RTI Routing Service Adapter SDK*; it must be installed on top of *RTI Routing Service*.

This adapter can read and write from TCP sockets, serializing and deserializing *Connext DDS* DynamicData samples into a simple format.

The reader will establish a TCP connection and receive bytes, converting them to DynamicData samples for *Routing Service*. The writer will create a TCP server and forward samples provided by *Routing Service* to all the TCP clients that connect to it.

This adapter is similar to the file adapter in Example 9 - Using a File Adapter (Section 3.9); the following instructions assume that you have gone through that example.

Compile the adapter:

1. Compile the socket adapter in **adapters/socket**:
   - On UNIX-based systems:
     - Set the NDDSHOME environment variable as described in the *RTI Connext DDS Core Libraries Getting Started Guide*.
     - Enter:

       ```
       cd <path to examples>/routing_service/adapters/socket/make
       gmake -f Makefile.<architecture>
       ```

       The adapter shared library, **libsocketadapter.so**, will be copied to **<path to examples>/routing_service/adapters/socket**.
   - On Windows systems:
     - Set the NDDSHOME environment variable as described in the *RTI Connext DDS Core Libraries Getting Started Guide*.
     - Open the Visual Studio solution under **<path to examples>\routing_service\adapters\socket\windows**. For example, if you are using Visual Studio 2013, open **socketadapter-vs2013.sln**.
     - Build the solution.

       The adapter shared library, **socketadapter.dll**, will be copied to **<path to examples>/routing_service/adapters/socket**.

Run the examples:

The example configuration file for *Routing Service* is located in **<path to examples>/routing_service/shapes/socket_bridge.xml**. It defines several <routing_service> configurations for different examples. We will use *Shapes Demo* to publish and subscribe to colored shapes. To send TCP traffic and listen to connections coming from the socket adapter, we will use the UNIX utility, netcat (**nc**).

Netcat is a UNIX utility with many different uses involving TCP or UDP. Among other things, it can open TCP connections and listen to arbitrary TCP ports and accept connections. We will use netcat in this example to communicate with the socket adapter. You could use your own socket application or any other third-party TCP utility of your choice.

2. Read from DDS and send through a TCP socket.
   a. Start *Shapes Demo* on domain 0 (the default domain) and publish some Squares.
   b. On machine HostA, run *Routing Service*:

```
cd <NDDSHOME>
bin/rtiroutingservice \
-cfgFile <path to examples>/routing_service/shapes/socket_bridge.xml \
-cfgName dds_to_socket
```

   c. On machine HostB, connect to HostA using netcat:

```
nc HostA 8112
```

   d. Data should be being printed in netcat.
   e. Optionally, run netcat as before on one or more additional machines. Data will be sent to all of them.
3. Read from a TCP socket and publish into DDS.
   a. The configuration file uses "MyHost" as a host name to connect to in different places. Find and replace them to use the host where you will run netcat (we will call it HostB).
   b. Start *Shapes Demo* on domain 0 (the default domain) and create a Square subscriber.
   c. On machine HostB, start netcat and listen to TCP connections on port 8111:

```
nc -l 8111
```

   d. On machine HostA, run *Routing Service*:

```
bin/rtiroutingservice \
-cfgFile <path to examples>/routing_service/shapes/socket_bridge.xml \
-cfgName socket_to_dds
```

   e. From netcat (HostB), enter:

```
x=100,y=100,shapesize=50,color=RED;
x=150,y=150,shapesize=10,color=BLUE;
```

   You should see two squares in *Shapes Demo.*
4. Use remote administration to change the host *Routing Service* is writing to.
   a. On another machine, HostC run netcat:

```
nc -l 8111
```

   b. On any machine, using the *Routing Service* shell (see Section 3.6) on domain 1, send the following command (all on one line):

```
RTI Routing Service> update socket_to_dds
socketdds::s::square_socket route.input.property[Host]=HostC
```

   c. From netcat (HostC), enter:

```
x=100,y=100,shapesize=10,color=RED;
x=150,y=150,shapesize=50,color=BLUE;
```

d. In *Shapes Demo*, the size of the shapes should have changed.

e. The *Routing Service* adapter API lets an adapter update its configuration on the fly when the service receives a remote command that changes its configuration properties. Explore the code to see how **RTI_RoutingServiceSocketStreamReader_update()** is implemented.

5. There are two other <routing_service> tags in the configuration file. One of them (socket_to_socket) eliminates DDS from the picture and just reads from a socket and writes to a different one. The other one, socket_tester, reads from TCP and uses a Java adapter that works as an output test adapter, counting the samples received and reporting failure or success depending the expected result configured with the tag <property>.

Customize the Socket Adapter

In the example, the socket adapters use a specific format, which you already saw in netcat. Now try adapting the example to your own format. The code that serializes/deserializes DynamicData samples to/from a byte buffer is in **adapters/socket/src/SampleParsing.c**.

6. Edit the function **RTI_RoutingServiceSocketAdapter_parse_sample()** to implement how a DynamicData sample gets created from a byte buffer

7. In the same file, edit the function **RTI_RoutingServiceSocketAdapter_serialize_sample()** to implement how a sample is converted to a byte buffer.

8. Compile the code as described in Step 1 on page 3-16.