RTI Connext

Prototyper

Getting Started Guide

Version 5.0.0

Your systems. Working as one

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

1 Introduction

RTI® Connext™ Prototyper is a tool to accelerate *RTI Connext* application development and scenario testing. It provides *Connext* application developers with a quick and easy-to-use mechanism to try out realistic scenarios on their own computer systems and networks, and get immediate information on the expected performance, resource usage, and behavior of their system.

With the traditional approach, if you want to try a specific *Connext* distributed application design and determine Key Performance Indicators (KPIs), you would have to spend significant time and effort to develop a custom prototype that could determine KPIs such as:

- The memory a particular application is likely to use.
- The time it will take for discovery to complete.
- The system bandwidth the running application will consume.
- The CPU usage it will take for a particular application to publish its data at a certain rate, or to receive a certain set of Topics.
- The impact of changing data types, topics, Quality of Service, and other design parameters.

Prototyper significantly simplifies this process. Instead of writing custom code, you can:

1. Describe the system in an XML file (or files),
2. Run *Prototyper* on each computer, specifying the particular configuration for that computer, and
3. Observe the behavior of the running system and read the KPIs from the *RTI Monitor* tool.

This document assumes you have a basic understanding of Connext application development and concepts such as a DDS Domain, DomainParticipant, Topic, DataWriter and DataReader. For an overview of these concepts, please see the RTI Core Libraries and Utilities Getting Started Guide (<Connext installation directory>/ndds.<version>/doc/PDF/RTI_CoreLibrariesAndUtilities_GettingStarted.pdf).

Prototyper is a command-line executable application. Once installed, *Prototyper* can be found in the **scripts**[1] directory. *Prototyper* takes several command-line parameters, which allow you to specify the XML configuration file, the specific *DomainParticipant* configuration to use, and other run-time parameters such as the rate at which to publish data. You must start *Prototyper* manually on each machine where you would like to run it, specifying the appropriate parameters.

The XML file-format used by *Prototyper* is compatible with the one used for the Connext XML-Based Application Creation feature. This means that your investment in describing your system via XML can be fully leveraged during your application development. That is, the *data-types*, *Topics*, *DomainParticipants,* and other entities described in the XML file can be directly created from application code and integrated into your final application without the need to re-code them in the source files. Please refer to the *XML-Based Application Creation Getting Started Guide* (RTI_CoreLibrariesAndUtilities_XML_AppCreation_GettingStarted.pdf) for a description of this feature and the format used to describe Connext applications in XML.

[1] <*Connext installation directory*>/ndds.<version>/scripts

2 “Hello World” with RTI Connext Prototyper

This section assumes you have installed the software and configured your environment correctly. If you have not done so, please follow the steps in the RTI Core Libraries and Utilities Getting Started Guide, specifically Chapter 2 “Installing Connext” and Section 3.1 “Building and Running Hello World”. This guide is part of your distribution (RTI_CoreLibrariesAndUtilities_GettingStarted.pdf). The guide will assist you in correctly setting both your environment variable ***NDDSHOME*** *and, depending on your architecture, the environment variable PATH (on Windows Systems), LD_LIBRARY_PATH (on Linux systems), or DYLD_LIBRARY_PATH (on MacOS Systems).*

Before you begin, please verify that your NDDSHOME environment variable is set correctly. It should point to a directory named **ndds.5.0.x** (where ‘*x*’ is a release-specific number for your distribution) within your *Connext* installation.

To verify that the variable NDDSHOME is set correctly, open a UNIX command shell or a Windows prompt and follow the steps below.

On a Windows system:

Open a command prompt (from the Start menu, select **All Programs, Accessories, Command Prompt**) and execute the following command:

```
type “%NDDSHOME%”\rev*
```

If NDDSHOME is set correctly, you will see an output similar to this:

```
Host Build 5.0.0 rev 00 (0x05000000)
Host Target 5.0.0 rev 00 (0x05000000)
```

If it is not set correctly, you will see something like:

```
The system cannot find the path specified.
```

Or something like:

```
The system cannot find the file specified.
```

On a UNIX-based system:

Execute the following command in a shell window:

```
cat $NDDSHOME/rev*
```

If NDDSHOME is set correctly, you will see an output similar to this:

```
Host Build 5.0.0 rev 00 (0x05000000)
Host Target 5.0.0 rev 00 (0x05000000)
```

If it is not set correctly, you will see something like:

```
cat: /rev*: No such file or directory
```

If NDDSHOME is not set correctly, please follow the steps in the *RTI Core Libraries and Utilities Getting Started Guide* mentioned above in order to set it.

2.1 Hello World Prototype

The files for this example are located in the directory:

- $NDDSHOME/example/prototyper/HelloWorld on UNIX-based systems
- %NDDSHOME%\example\prototyper\HelloWorld on Windows systems

This simple scenario defines two *DomainParticipant* configurations, illustrated in the figure below: "PublicationParticipant" which writes to the Topic "HelloWorldTopic" and "SubscriptionParticipant" which subscribes to that Topic.

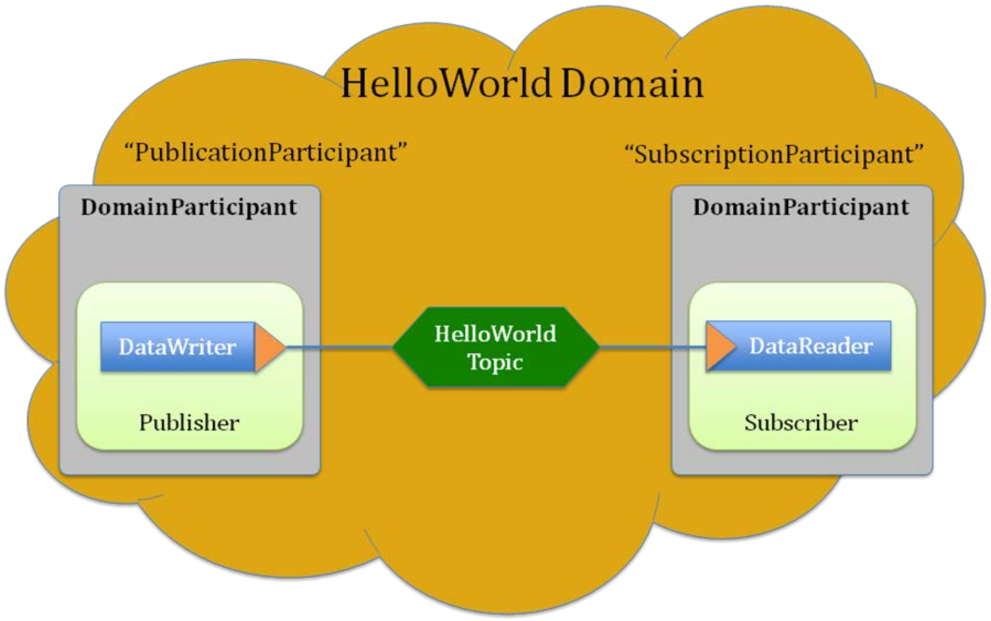


Figure 1: The XML file defines two DomainParticipant configurations: "PublicationParticipant" and "SubscriptionParticipant"

First, we will run the scenario. Then we will examine the configuration files.

2.1.1 Run Prototyper

On a Windows system:

Open two command-prompt windows. In each one, **change the directory** to **%NDDSHOME%\example\prototyper\HelloWorld** and type the following command in each window:

```
..\..\..\scripts\rtiddsprototyper.bat
```

On a UNIX-based system:

Open two command-shell windows. In each one, **change the directory** to **$NDDSHOME/example/prototyper/HelloWorld** and type the following command in each window:

```
../../../scripts/rtiddsprototyper
```

You will see the following output appear on each window:

```
Please select among the available configurations:
0: MyParticipantLibrary::PublicationParticipant
1: MyParticipantLibrary::SubscriptionParticipant
Please select:
```

If you do not see this output and get the following error instead:

```
rtidsprototyper: Error configuration file not found.
```

This indicates that you did not run **rtiddsprototyper** from the right directory. Please change directories to **%NDDSHOME%\example\prototyper\HelloWorld** on Windows systems, or **$NDDSHOME/example/prototyper/HelloWorld** on UNIX-based systems directory and verify you see the file USER_QOS_PROFILES.xml in that directory.

In one of the windows type "0" (without the quotes) to select the first choice, followed by a return. In the other window, type "1" (without the quotes) to select the second choice, also followed by a return.

In the window where you typed "0" (first choice), you will see output like this:

```
Please select among the available configurations:
0: MyParticipantLibrary::PublicationParticipant
1: MyParticipantLibrary::SubscriptionParticipant
Please select: 0
DataWriter "HelloWorldWriter" wrote sample 1 on Topic "HelloWorldTopic" at 1332618800.504111 s
DataWriter "HelloWorldWriter" wrote sample 2 on Topic "HelloWorldTopic" at 1332618801.504341 s
DataWriter "HelloWorldWriter" wrote sample 3 on Topic "HelloWorldTopic" at 1332618802.504593 s
```

In the window where you typed "1" (second choice), you will see output like this:

```
Please select among the available configurations:
0: MyParticipantLibrary::PublicationParticipant
1: MyParticipantLibrary::SubscriptionParticipant
Please select: 1
DataReader "HelloWorldReader" received sample 1 on Topic "HelloWorldTopic" sent at
1332618800.504111 s
sender: "Key: 521035021"
message: "String: 1"
count: 1

DataReader "HelloWorldReader" received sample 2 on Topic "HelloWorldTopic" sent at
1332618801.504341 s
sender: "Key: 521035021"
message: "String: 2"
count: 2

DataReader "HelloWorldReader" received sample 3 on Topic "HelloWorldTopic" sent at
1332618802.504593 s
sender: "Key: 521035021"
message: "String: 3"
count: 3
```

2.1.2 Examine the XML Configuration File

Let's review the content of the file "USER_QOS_PROFILES.xml" in the **example/prototyper/HelloWorld** directory.

```xml
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="../../../resource/schema/rti_dds_profiles.xsd"
     version="5.0.0">

    <!-- Qos Library -->
    <qos_library name="qosLibrary">
        <qos_profile name="TransientDurability"
                     is_default_qos="true">
            <datawriter_qos>
                <durability>
                    <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
                </durability>
                <reliability>
                    <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <history>
                    <kind>KEEP_LAST_HISTORY_QOS</kind>
                    <depth>20</depth>
                </history>
            </datawriter_qos>
            <datareader_qos>
                <durability>
                    <kind>TRANSIENT_LOCAL_DURABILITY_QOS</kind>
                </durability>
                <reliability>
                    <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <history>
                    <kind>KEEP_LAST_HISTORY_QOS</kind>
                    <depth>10</depth>
                </history>
            </datareader_qos>
        </qos_profile>
    </qos_library>

    <!-- types -->
    <types>
        <const name="MAX_NAME_LEN" type="long" value="64"/>
        <const name="MAX_MSG_LEN"  type="long" value="128"/>

        <struct name="HelloWorld">
            <member name="sender"  key="true"
                    type="string"  stringMaxLength="MAX_NAME_LEN"/>
            <member name="message"
```

```
                        type="string"  stringMaxLength="MAX_MSG_LEN"/>
                <member name="count"   type="long"/>
             </struct>
       </types>

       <!-- Domain Library -->
       <domain_library name="MyDomainLibrary" >

           <domain name="HelloWorldDomain" domain_id="0">
               <register_type name="HelloWorldType" kind="dynamicData"
                              type_ref="HelloWorld" />

               <topic name="HelloWorldTopic"
                      register_type_ref="HelloWorldType"/>
           </domain>
       </domain_library>

       <!-- Participant library -->
       <participant_library name="MyParticipantLibrary">

           <domain_participant name="PublicationParticipant"
                         domain_ref="MyDomainLibrary::HelloWorldDomain">

               <publisher name="MyPublisher">
                   <data_writer name="HelloWorldWriter"
                                topic_ref="HelloWorldTopic">
                       <datawriter_qos name="HelloWorld_writer_qos"
                            base_name="qosLibrary::TransientDurability"/>
                   </data_writer>
               </publisher>
           </domain_participant>

            <domain_participant name="SubscriptionParticipant"
                         domain_ref="MyDomainLibrary::HelloWorldDomain">


               <subscriber name="MySubscriber">

                   <data_reader name="HelloWorldReader"
                                topic_ref="HelloWorldTopic">
                       <datareader_qos name="HelloWorld_reader_qos"
                            base_name="qosLibrary::TransientDurability"/>
                   </data_reader>
               </subscriber>
           </domain_participant>
```

```

    </participant_library>

</dds>
```

The configuration file contains four main sections:

- QoS definition section (**<qos_library>** tag).
- Type definition section (**<types>** tag).
- Domain definition section (**<domain_library>** tag).
- Participant definition section (**<participant_library>** tag).

> The structure and syntax of the XML configuration file is identical to the one used for XML Application Creation. Please refer to *XML-Based Application Creation Getting Started Guide* for a detailed description of the format of the XML configuration file.

Examining the file we can see that it defines:

- A QoS library "qosLibrary" containing a single QoS Profile "TransientDurability."
- A data type "HelloWorld" with members 'sender', 'message', and 'count'.
- A domain library "MyDomainLibrary" containing a single domain "HelloWorldDomain" with topic "HelloWorldTopic."
- A *DomainParticipant* library "MyParticipantLibrary" containing two **DomainParticipant configurations**: "PublicationParticipant" and "SubscriptionParticipant":
    - The "PublicationParticipant" publishes the "HelloWorldTopic"
    - The "SubscriptionParticipant" subscribes to the "HelloWorldTopic"

These definitions correspond to the distributed application shown in Figure 1.

2.1.3 Behavior of Prototyper for the HelloWorld Application

Prototyper gets its configuration from a set of XML files. By default, *Prototyper* will look in the current working directory for a file called "USER_QOS_PROFILES.xml" and read it to determine the defined participant configurations and offer them as choices.

In this example, *Prototyper* found two participant configurations and offered them as choices on the command line: "MyParticipantLibrary::PublicationParticipant" and "MyParticipantLibrary::SubscriptionParticipant".

You can control this behavior via the command-line options so that *Prototyper* reads a different file and/or automatically starts a particular participant configuration. For example, you can type this on the command line to start the "MyParticipantLibrary::PublicationParticipant":

On a Windows system:

```
..\..\..\scripts\rtiddsprototyper -cfgName "MyParticipantLibrary::PublicationParticipant"
```

On a UNIX-based system:

```
../../../scripts/rtiddsprototyper -cfgName "MyParticipantLibrary::PublicationParticipant"
```

Please see Section 3 for more details on the behavior of *Prototyper* and the command-line options it accepts.

2.2 Shapes Demo Prototype

The files for this example are in the directory:

- $NDDSHOME/example/prototyper/Shapes on UNIX-based systems
- %NDDSHOME%\example\prototyper\Shapes on Windows systems

This scenario defines three participant configurations, illustrated in the figure below: "ShapePublisher" which writes to the Topics "Square" and "Circle"; "ShapeSubscriber" which subscribes to the Topics "Square", "Circle", and "Triangle"; and "ShapePubSub" which publishes "Triangle" and subscribes to "Circle". This system is illustrated in Figure 2. The DDS *domain* is defined with the same Topics and data-types used by *RTI Shapes Demo* such that it can be used in conjunction with it.

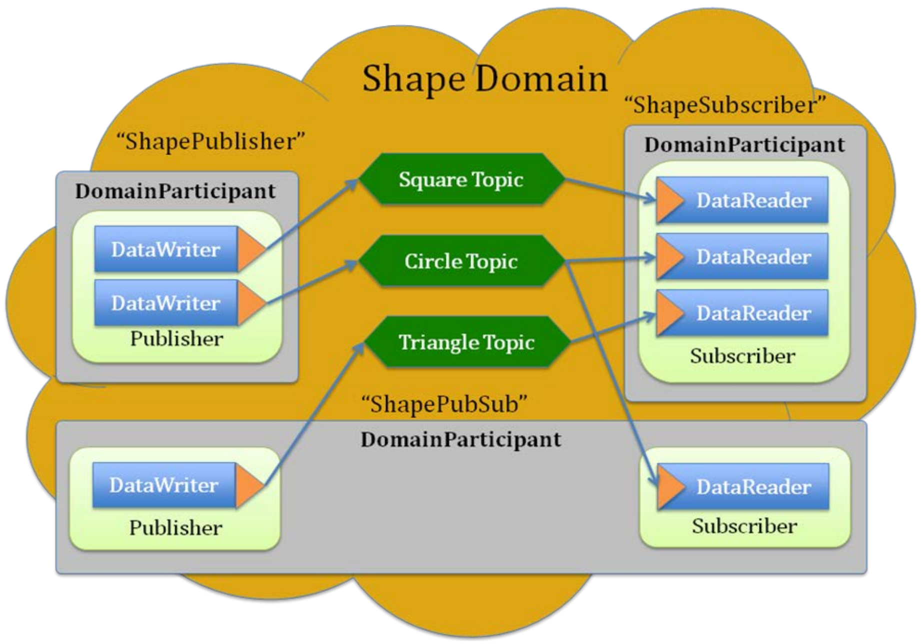


Figure 2: Shape Domain contains three participant configurations: ShapePublisher, ShapeSubscriber, and ShapePubSub

2.2.1 Run Prototyper

On a Windows system:

Open three command-prompt windows. In each one, change the directory to **example\prototyper\Shapes** and type the following command in each window:

```
..\..\..\scripts\rtiddsprototyper.bat
```

On a UNIX-based system:

Open two command-shell windows. In each one, change the directory to **example/prototyper/Shapes** and type the following command in each window:

```
../../../scripts/rtiddsprototyper
```

You will see the following output appear on each window:

```
Please select among the available configurations:
0: MyParticipantLibrary::ShapePublisher
```

```
  1: MyParticipantLibrary::ShapeSubscriber
  2: MyParticipantLibrary::ShapePubSub
  Please select:
```

If you do not see this output and get the following error instead:

```
rtidsprototyper: Error configuration file not found.
```

It indicates that you did not run rtiddsprototyper from the right directory. Please change directories to the **example/prototyper/Shapes** directory and make sure you see a file named USER_QOS_PROFILES.xml.

In the first window, type "0" (without the quotes) to select the first choice, followed by a return. In the second, type "1" (without the quotes) to select the second choice, also followed by a return. In the third window, type "2".

In the window where you typed "0" (first choice), you should see output like this:

```
Please select among the available configurations:
0: MyParticipantLibrary::ShapePublisher
1: MyParticipantLibrary::ShapeSubscriber
2: MyParticipantLibrary::ShapePubSub
Please select: 0

  DataWriter "MySquareWriter" wrote sample 1 on Topic "Square" at 1332619432.759611 s
  DataWriter "MyCircleWriter" wrote sample 1 on Topic "Circle" at 1332619432.759720 s
  DataWriter "MySquareWriter" wrote sample 2 on Topic "Square" at 1332619433.759838 s
  DataWriter "MyCircleWriter" wrote sample 2 on Topic "Circle" at 1332619433.759953 s
  DataWriter "MySquareWriter" wrote sample 3 on Topic "Square" at 1332619434.760090 s
  DataWriter "MyCircleWriter" wrote sample 3 on Topic "Circle" at 1332619434.760202 s
  DataWriter "MySquareWriter" wrote sample 4 on Topic "Square" at 1332619435.760281 s
  DataWriter "MyCircleWriter" wrote sample 4 on Topic "Circle" at 1332619435.760432 s
  DataWriter "MySquareWriter" wrote sample 5 on Topic "Square" at 1332619436.760471 s
  DataWriter "MyCircleWriter" wrote sample 5 on Topic "Circle" at 1332619436.760591 s
  DataWriter "MySquareWriter" wrote sample 6 on Topic "Square" at 1332619437.760687 s
  DataWriter "MyCircleWriter" wrote sample 6 on Topic "Circle" at 1332619437.760819 s
  DataWriter "MySquareWriter" wrote sample 7 on Topic "Square" at 1332619438.760921 s
  DataWriter "MyCircleWriter" wrote sample 7 on Topic "Circle" at 1332619438.761073 s
```

We can see it has two writers "MySquareWriter" and "MyCircleWriter" and we should also see how at the periodic rate it writes two samples, one on each *DataWriter*. This is because the "ShapePublisher" configuration specified two writers: one for "Square" and one for "Circle".

In the window where you typed "1" (second choice), you should see output similar to this:

```
Please select among the available configurations:
Please select among the available configurations:
0: MyParticipantLibrary::ShapePublisher
1: MyParticipantLibrary::ShapeSubscriber
2: MyParticipantLibrary::ShapePubSub
Please select: 1
DataReader "MySquareRdr" received sample 4 on Topic "Square" sent at 1332619435.760281 s
color: "Key: 628974580"
x: 4
y: 4
shapesize: 4
```

```
DataReader "MyCircleRdr" received sample 4 on Topic "Circle" sent at 1332619435.760432 s
color: "Key: 1894519218"
x: 4
y: 4
shapesize: 4

DataReader "MySquareRdr" received sample 5 on Topic "Square" sent at 1332619436.760471 s
color: "Key: 628974580"
x: 5
y: 5
shapesize: 5

DataReader "MyCircleRdr" received sample 5 on Topic "Circle" sent at 1332619436.760591 s
color: "Key: 1894519218"
x: 5
y: 5
shapesize: 5

DataReader "MyTriangleRdr" received sample 2 on Topic "Triangle" sent at 1332619437.609176 s
color: "Key: 333582338"
x: 2
y: 2
shapesize: 2

DataReader "MySquareRdr" received sample 6 on Topic "Square" sent at 1332619437.760687 s
color: "Key: 628974580"
x: 6
y: 6
shapesize: 6
DataReader "MyCircleRdr" received sample 6 on Topic "Circle" sent at 1332619437.760819 s
color: "Key: 1894519218"
x: 6
y: 6
shapesize: 6

DataReader "MyTriangleRdr" received sample 3 on Topic "Triangle" sent at 1332619438.609384 s
color: "Key: 333582338"
x: 3
y: 3
shapesize: 3

DataReader "MySquareRdr" received sample 7 on Topic "Square" sent at 1332619438.760921 s
color: "Key: 628974580"
x: 7
y: 7
shapesize: 7

DataReader "MyCircleRdr" received sample 7 on Topic "Circle" sent at 1332619438.761073 s
color: "Key: 1894519218"
x: 7
y: 7
shapesize: 7

DataReader "MyTriangleRdr" received sample 4 on Topic "Triangle" sent at 1332619439.609556 s
color: "Key: 333582338"
x: 4
y: 4
shapesize: 4
```

We see that initially it is receiving samples “Key: 628974580” of Topic “Square” and “Key: 1894519218” of Topic “Circle”. After a while, it also starts receiving samples with “Key: 333582338” of Topic “Triangle”.

Note that depending on the relative timing when your applications start, the results you see may differ from this.

The reason for this is that the “ShapeSubscriber” configuration subscribes to “Square”, “Circle”, and “Triangle”. Initially we had only started the “ShapePublisher” configuration, which just publishes samples “Key: 628974580” on Topic “Square” and “Key: 1894519218” on the Topic “Circle”. After a little while, we started the “ShapePubSub” configuration which publishes samples of the Topic “Triangle” with the key “Key: 333582338”.

In the window where you typed “2” (third choice), you should see output similar to this:

```
Please select among the available configurations:
0: MyParticipantLibrary::ShapePublisher
1: MyParticipantLibrary::ShapeSubscriber
2: MyParticipantLibrary::ShapePubSub
Please select: 2
DataWriter "MyTriangleWr" wrote sample 1 on Topic "Triangle" at 1332619436.608954 s
DataReader "MyCircleRdr" received sample 5 on Topic "Circle" sent at 1332619436.760591 s
color: "Key: 1894519218"
x: 5
y: 5
shapesize: 5

DataWriter "MyTriangleWr" wrote sample 2 on Topic "Triangle" at 1332619437.609176 s
DataReader "MyCircleRdr" received sample 6 on Topic "Circle" sent at 1332619437.760819 s
color: "Key: 1894519218"
x: 6
y: 6
shapesize: 6

DataWriter "MyTriangleWr" wrote sample 3 on Topic "Triangle" at 1332619438.609384 s
DataReader "MyCircleRdr" received sample 7 on Topic "Circle" sent at 1332619438.761073 s
color: "Key: 1894519218"
x: 7
y: 7
shapesize: 7
```

We initially see that it is both writing data on the Topic “Triangle” and receiving data on the Topic “Circle”. The only values on Topic “Circle” are the ones from the “ShapePublisher,” which is only writing samples with the key “Key: 1894519218”.

Depending on the relative timing in which you started your applications your results may differ from these.

If you look carefully at the output of the “ShapeSubscriber” and “ShapePubSub” configurations, you may notice that they do not receive the first samples that are published by the “ShapePublisher” configuration. You should not be too concerned about this. It is because the default Quality of Service (QoS) settings used in this scenario specify that the data should only be sent to the readers that are present at the time the data is sent (this is known as VOLATILE Durability). It can be easily changed; that is in fact what the QoS profile used in the HelloWorld example in Section 2.1 did.

2.2.2 Examine the XML Configuration File

Let’s review the content of “USER_QOS_PROFILES.xml” in the example/prototyper/Shapes directory.

```xml
<!--
RTI Data Distribution Service Deployment
-->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="../../../resource/schema/rti_dds_profiles.xsd"
     version="5.0.0">

    <!-- Qos Library -->
    <qos_library name="qosLibrary">
        <qos_profile name="defaultProfile" is_default_qos="true">
        </qos_profile>
    </qos_library>

    <!-- types -->
    <types>
         <const name="MAX_COLOR_LEN" type="long" value="32"/>

         <struct name="ShapeType">
             <member name="color"  key="true"
                     type="string"  stringMaxLength="MAX_COLOR_LEN"/>
             <member name="x" type="long"/>
             <member name="y" type="long"/>
             <member name="shapesize" type="long"/>
         </struct>
    </types>

    <!-- Domain Library -->
    <domain_library name="MyDomainLibrary" >

        <domain name="ShapeDomain" domain_id="0">
            <register_type name="ShapeType" kind="dynamicData"
                           type_ref="ShapeType" />

            <topic name="Square"   register_type_ref="ShapeType"/>
            <topic name="Circle"   register_type_ref="ShapeType"/>
            <topic name="Triangle" register_type_ref="ShapeType"/>

        </domain>
    </domain_library>

    <!-- Participant library -->
    <participant_library name="MyParticipantLibrary">

        <!-- 1st participant: publishes Square and Circle
        -->
        <domain_participant name="ShapePublisher"
                            domain_ref="MyDomainLibrary::ShapeDomain">

            <publisher name="MyPublisher">
                <data_writer name="MySquareWriter" topic_ref="Square"/>
                <data_writer name="MyCircleWriter" topic_ref="Circle"/>
            </publisher>
        </domain_participant>

        <!-- 2nd participant: subscribes Square, Circle, and Triangle
        -->
        <domain_participant name="ShapeSubscriber"
                             domain_ref="MyDomainLibrary::ShapeDomain">

            <subscriber name="MySubscriber">
                <data_reader name="MySquareRdr"   topic_ref="Square"/>
                <data_reader name="MyCircleRdr"   topic_ref="Circle"/>
                <data_reader name="MyTriangleRdr" topic_ref="Triangle"/>
            </subscriber>
        </domain_participant>

        <!-- 3rd participant: publishes Triangle and subscribes Circle
        -->
        <domain_participant name="ShapePubSub"
                             domain_ref="MyDomainLibrary::ShapeDomain">

            <publisher name="MyPublisher">
                <data_writer name="MyTriangleWr" topic_ref="Triangle"/>
            </publisher>

            <subscriber name="MySubscriber">
                <data_reader name="MyCircleRdr"  topic_ref="Circle"/>
            </subscriber>
        </domain_participant>

    </participant_library>
</dds>
```

Similar to what we saw in the HelloWorld example, the configuration file contains four main sections:

- QoS definition section (**<qos_library>** tag).
- Type definition section (**<types>** tag).
- Domain definition section (**<domain>** tag).
- Participant definition section (**<participant_library>** tag).

The structure and syntax of the XML configuration file is identical to the one used for XML Application Creation. Please refer to *XML-Based Application Creation Getting Started* Guide for a detailed description of the format of the XML configuration file.

Examining the file, we can see that it defines:

- A QoS library "qosLibrary" containing a single QoS Profile "defaultProfile ."
- A data type "ShapeType" with fields 'color', 'x', 'y', and 'shapesize'.
- A domain library "MyDomainLibrary" containing a single domain "ShapeDomain" with topics "Square", "Circle", and "Triangle". All these topics use the same registered data type "ShapeType".
- A domain participant library "MyParticipantLibrary" containing three **DomainParticipant configurations**: "ShapePublisher", "ShapeSubscriber", and "ShapePubSub."
    - The "ShapePublisher" configuration publishes the topics "Square" and "Circle.
    - The "ShapeSubscriber" configuration subscribes to topics "Square", "Circle", and "Triangle"
    - The "ShapePubSub" configuration publishes topic "Triangle" and subscribes to topic "Circle".

These definitions correspond to the distributed application shown in Figure 2.

2.2.3 Run with the rtishapesdemo Application

Exit the three *Prototyper* applications started in Section 2.2.2 if they are still running. We will run them again, but this time we will use a different configuration file to control the data values *Prototyper* writes.

On a Windows system:

Open three command-shell (windows terminal) windows. In each window, change the directory to **%NDDSHOME%\example\prototyper\Shapes** and type the following command in each window:

```
..\..\..\scripts\rtiddsprototyper.bat –cfgFile ShapeDemoConfig.xml
```

On a UNIX-based system:

Open two command-shell windows. In each one, change the directory to **$NDDSHOME/example/prototyper/Shapes** and type the following command in each window:

```
../../../scripts/rtiddsprototyper  –cfgFile ShapeDemoConfig.xml
```

The last argument, following the "-cfgFile" directive, specifies an additional configuration file to read.

You will see the following output appear in each window:

```
Please select among the available configurations:
0: MyParticipantLibrary::ShapePublisher
1: MyParticipantLibrary::ShapeSubscriber
2: MyParticipantLibrary::ShapePubSub
3: MyParticipantLibrary::ControlledShapePublisher
4: MyParticipantLibrary::ControlledShapeSubscriber
5: MyParticipantLibrary::ControlledShapePubSub
Please select:
```

We see three more configurations (index 3 to 5) beyond the ones we get if we do not specify the "–cfgFile ShapeDemoConfig.xml" directive. These additional configurations have "Controlled" in their name.

The additional configurations are the result of reading the configuration file "ShapeDemoConfig.xml" specified in the command line. The participant configurations defined in the "USER_QOS_PROFILES.xml" file still appear. This is because the file USER_QOS_PROFILE.xml is also being read. This is desirable; as we will see later the new configurations defined in ShapeDemoConfig.xml are using information that was defined in USER_QOS_PROFILE.xml

1. In the first window, type "3" (without the quotes) to select the first choice, followed by a return.
2. In the second type "4" (without the quotes) to select the second choice, also followed by a return.
3. In the third window type "5" (without the quotes).

In the window where you typed "3", you will see output like this:

```
Please select among the available configurations:
0: ParticipantLibrary::ShapePublisher
1: ParticipantLibrary::ShapeSubscriber
2: ParticipantLibrary::ShapePubSub
3: ParticipantLibrary::ControlledShapePublisher
4: ParticipantLibrary::ControlledShapeSubscriber
5: ParticipantLibrary::ControlledShapePubSub
Please select: 3
DataWriter "MySquareWriter" wrote sample 1 on Topic "Square" at 1332634406.819248 s
DataWriter "MyCircleWriter" wrote sample 1 on Topic "Circle" at 1332634406.819343 s
DataWriter "MySquareWriter" wrote sample 2 on Topic "Square" at 1332634407.819631 s
DataWriter "MyCircleWriter" wrote sample 2 on Topic "Circle" at 1332634407.819794 s
DataWriter "MySquareWriter" wrote sample 3 on Topic "Square" at 1332634408.819853 s
DataWriter "MyCircleWriter" wrote sample 3 on Topic "Circle" at 1332634408.819977 s
DataWriter "MySquareWriter" wrote sample 4 on Topic "Square" at 1332634409.820010 s
DataWriter "MyCircleWriter" wrote sample 4 on Topic "Circle" at 1332634409.820189 s
DataWriter "MySquareWriter" wrote sample 5 on Topic "Square" at 1332634410.820311 s
DataWriter "MyCircleWriter" wrote sample 5 on Topic "Circle" at 1332634410.820490 s
DataWriter "MySquareWriter" wrote sample 6 on Topic "Square" at 1332634411.820494 s
DataWriter "MyCircleWriter" wrote sample 6 on Topic "Circle" at 1332634411.820686 s
DataWriter "MySquareWriter" wrote sample 7 on Topic "Square" at 1332634412.820663 s
DataWriter "MyCircleWriter" wrote sample 7 on Topic "Circle" at 1332634412.820845 s
DataWriter "MySquareWriter" wrote sample 8 on Topic "Square" at 1332634413.820869 s
DataWriter "MyCircleWriter" wrote sample 8 on Topic "Circle" at 1332634413.821030 s
DataWriter "MySquareWriter" wrote sample 9 on Topic "Square" at 1332634414.821189 s
DataWriter "MyCircleWriter" wrote sample 9 on Topic "Circle" at 1332634414.821348 s
```

We can see it has two writers: “MySquareWriter” and “MyCircleWriter” We should also see how at the periodic rate, it writes two samples, one on each *DataWriter*. This is because the “ShapePublisher” configuration specified two writers: one for “Square” and one for “Circle”.

In the window where you typed “4”, you will see output like this:

```
Please select among the available configurations:
0: ParticipantLibrary::ShapePublisher
1: ParticipantLibrary::ShapeSubscriber
2: ParticipantLibrary::ShapePubSub
3: ParticipantLibrary::ControlledShapePublisher
4: ParticipantLibrary::ControlledShapeSubscriber
5: ParticipantLibrary::ControlledShapePubSub
Please select: 4
DataReader "MySquareRdr" received sample 4 on Topic "Square" sent at 1332634409.820010 s
color: "Red"
x: 3
y: 3
shapesize: 20

DataReader "MyCircleRdr" received sample 4 on Topic "Circle" sent at 1332634409.820189 s
color: "Orange"
x: 3
y: 153
shapesize: 30

DataReader "MySquareRdr" received sample 5 on Topic "Square" sent at 1332634410.820311 s
color: "Red"
x: 4
y: 4
shapesize: 20

DataReader "MyCircleRdr" received sample 5 on Topic "Circle" sent at 1332634410.820490 s
color: "Orange"
x: 4
y: 154
shapesize: 30

DataReader "MySquareRdr" received sample 6 on Topic "Square" sent at 1332634411.820494 s
color: "Red"
x: 5
y: 5
shapesize: 20

DataReader "MyCircleRdr" received sample 6 on Topic "Circle" sent at 1332634411.820686 s
color: "Orange"
x: 5
y: 155
shapesize: 30

DataReader "MyTriangleRdr" received sample 2 on Topic "Triangle" sent at 1332634412.308678 s
color: "Green"
x: 101
y: 1
shapesize: 30

DataReader "MySquareRdr" received sample 7 on Topic "Square" sent at 1332634412.820663 s
color: "Red"
x: 6
y: 6
```

```
shapesize: 20

DataReader "MyCircleRdr" received sample 7 on Topic "Circle" sent at 1332634412.820845 s
color: "Orange"
x: 6
y: 156
shapesize: 30

DataReader "MyTriangleRdr" received sample 3 on Topic "Triangle" sent at 1332634413.308962 s
color: "Yellow"
x: 102
y: 2
shapesize: 30

DataReader "MySquareRdr" received sample 8 on Topic "Square" sent at 1332634413.820869 s
color: "Red"
x: 7
y: 7
shapesize: 20

DataReader "MyCircleRdr" received sample 8 on Topic "Circle" sent at 1332634413.821030 s
color: "Orange"
x: 7
y: 157
shapesize: 30

DataReader "MyTriangleRdr" received sample 4 on Topic "Triangle" sent at 1332634414.309190 s
color: "Green"
x: 103
y: 3
shapesize: 30

DataReader "MySquareRdr" received sample 9 on Topic "Square" sent at 1332634414.821189 s
color: "Blue"
x: 8
y: 8
shapesize: 20

DataReader "MyCircleRdr" received sample 9 on Topic "Circle" sent at 1332634414.821348 s
color: "Orange"
x: 8
y: 158
shapesize: 30
```

The output is similar to what we saw in Section 2.2.1. The significant difference is the values taken by the data. Before, the values were assigned by a default internal algorithm that we could not control. Now, the data values are set according to a specification in ShapeDemoConfig.xml, which makes the values reasonable for the specific Shapes Demo application. We see ***color*** values like "Red", "Orange", "Green", *shapesize* values of 20 and 30, etc. We will examine the configuration that caused this to happen later in Section 2.2.4.

In the window where you typed "5", you will see output like this:

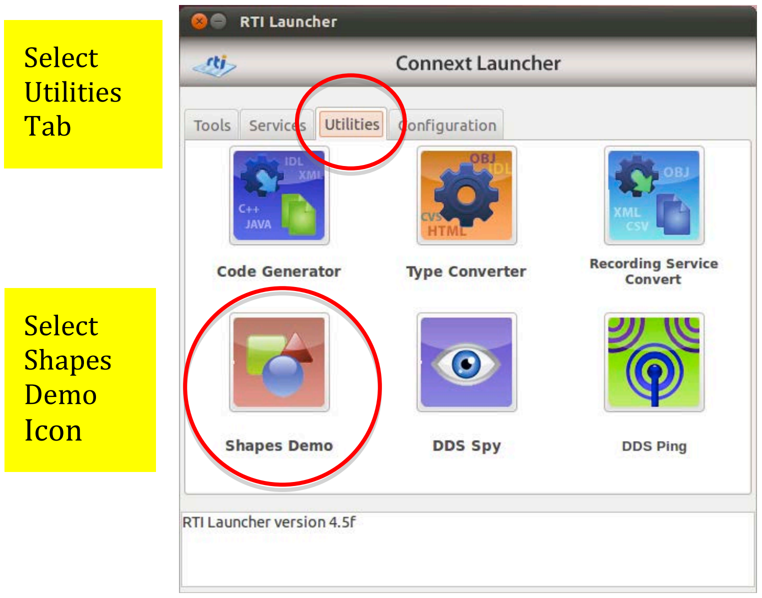
```
Please select among the available configurations:
0: ParticipantLibrary::ShapePublisher
1: ParticipantLibrary::ShapeSubscriber
2: ParticipantLibrary::ShapePubSub
3: ParticipantLibrary::ControlledShapePublisher
4: ParticipantLibrary::ControlledShapeSubscriber
5: ParticipantLibrary::ControlledShapePubSub
```

```
Please select: 5
DataWriter "MyTriangleWr" wrote sample 1 on Topic "Triangle" at 1332634411.308089 s
DataReader "MyCircleRdr" received sample 6 on Topic "Circle" sent at 1332634411.820686 s
color: "Orange"
x: 5
y: 155
shapesize: 30

DataWriter "MyTriangleWr" wrote sample 2 on Topic "Triangle" at 1332634412.308678 s
DataReader "MyCircleRdr" received sample 7 on Topic "Circle" sent at 1332634412.820845 s
color: "Orange"
x: 6
y: 156
shapesize: 30

DataWriter "MyTriangleWr" wrote sample 3 on Topic "Triangle" at 1332634413.308962 s
DataReader "MyCircleRdr" received sample 8 on Topic "Circle" sent at 1332634413.821030 s
color: "Orange"
x: 7
y: 157
shapesize: 30

DataWriter "MyTriangleWr" wrote sample 4 on Topic "Triangle" at 1332634414.309190 s
DataReader "MyCircleRdr" received sample 9 on Topic "Circle" sent at 1332634414.821348 s
color: "Orange"
x: 8
y: 158
shapesize: 30
```

The output is similar to what we saw in Section 2.2.1. The significant difference is in the values of the data itself, which as we will see below are now controlled via settings in ShapeDemoConfig.xml.

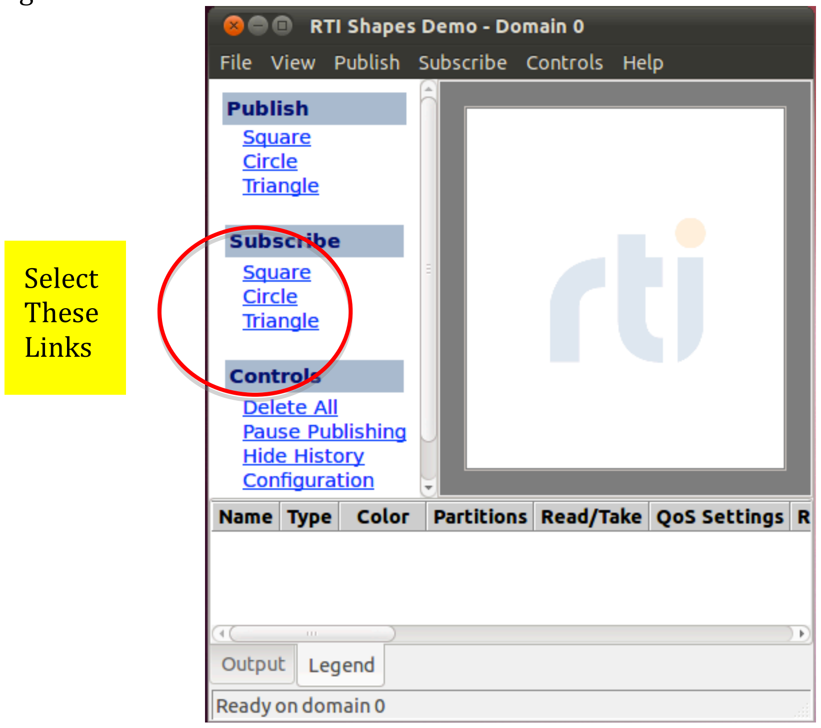
Run the RTI Shapes Demo Application

To start *RTI Shapes Demo,* open *RTI Connext Launcher*, select the "Utilities" tab and click on the "Shapes Demo" icon as indicated below.

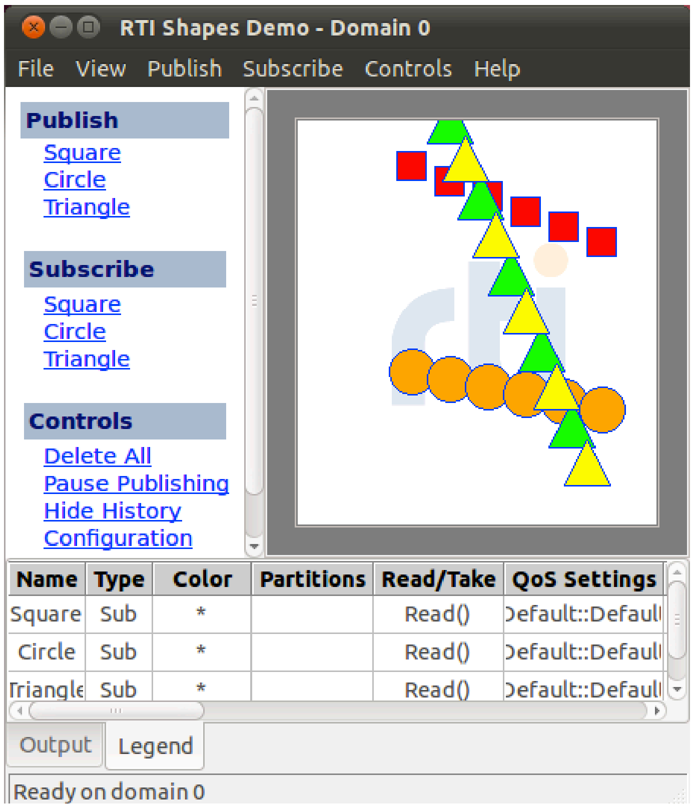


RTI Launcher version 4.5f

Once you have started *RTI Shapes Demo,* click on the left side links to subscribe to “Square”, “Circle” and “Triangle”.



You should immediately see the different shapes being drawn on the screen as the *RTI Shapes Demo* GUI receives them. This should look like the screenshot below:



Name	Type	Color	Partitions	Read/Take	QoS Settings
Square	Sub	*		Read()	Default::Defaul
Circle	Sub	*		Read()	Default::Defaul
Triangle	Sub	*		Read()	Default::Defaul

RTI Shapes Demo is receiving the same data as *Prototyper* when run with the "ControlledShapesSubscriber" configuration. Therefore we see updates for Topic "Square" with color "Red" (being published by the *Prototyper* with configuration "ControlledShapesPublisher"), updates for Topic "Circle" with color "Orange" " (also published by the *Prototyper* with configuration "ControlledShapesPublisher"), and updates for Topic "Triangle" with color alternating between "Green" and "Yellow".

We can also publish something from *Shapes Demo*, for example the Topic "Square" with color "BLUE" and we will immediately see it both in the Shapes Demo window (which subscribes to its own data) as well as the *Prototyper* that is running with the "ControlledShapesSubscriber" configuration. This is left as an exercise to the reader.

2.2.4 Behavior of Prototyper for the Shapes Demo Application

Prototyper reads its configuration from both the "USER_QOS_PROFILES.xml" and "ShapeDemoConfig.xml" files in the **$NDDSHOME/example/prototyper/Shapes** directory (**%NDDSHOME%\example\prototyper\Shapes** on Windows Systems). In these two files, *Prototyper* finds six participant configurations and offers these configurations as choices on the command line.
As an alternative, you can control this behavior using the command-line options so that *Prototyper* automatically starts with a particular participant configuration. For example, you can enter the following on the command line to create the *DomainParticipant* using the "MyParticipantLibrary::ShapePublisher" configuration:

```
../../../scripts/rtiddsprototyper –cfgName "MyParticipantLibrary::ShapePublisher"
```

The participant configurations that we used to interact with *RTI Shapes Demo* are defined in the "ShapeDemoConfig.xml" file. The configurations in this file control the values written by the *DataWriters* in a specific way to make them better suited to the scenario being run. For example, we see that the **color** members are set to reasonable values ("Red", "Orange", Green", "Yellow"). The values for the other members such as x, y, and shapesize, are also set in a way that allows Shapes Demo to display the data. To see how this is done, let's review the content of ShapeDemoConfig.xml, found in the directory:

- **$NDDSHOME/example/prototyper/Shapes** on UNIX-based systems
- **%NDDSHOME%\example\prototyper\Shapes** on Windows systems

```
<!--
RTI Data Distribution Service Deployment
-->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="../../../resource/schema/rti_dds_profiles.
xsd"
     version="5.0.0">

    <!-- Participant library -->
    <participant_library name="MyParticipantLibrary">

        <domain_participant name="ControlledShapePublisher"
                            domain_ref="MyDomainLibrary::ShapeDomain">

```

```
            <publisher name="MyPublisher">
                <data_writer name="MySquareWriter" topic_ref="Square">
                    <datawriter_qos>
                      <property>
                        <value>
                          <element>
                            <name>rti.prototyper.member:color</name>
                            <value>iterator?list=[Red]</value>
                          </element>
                          <element>
                            <name>rti.prototyper.member:x</name>
                            <value>linear?begin=0,end=100</value>
                          </element>
                          <element>
                            <name>rti.prototyper.member:y</name>
                            <value>linear?begin=0,end=100</value>
                          </element>
                          <element>
                            <name>rti.prototyper.member:shapesize</name>
                            <value>linear?begin=20,end=20</value>
                          </element>
                        </value>
                      </property>
                    </datawriter_qos>
                </data_writer>
                <data_writer name="MyCircleWriter" topic_ref="Circle">
                    <datawriter_qos>
                      <property>
                        <value>
                          <element>
                            <name>rti.prototyper.member:color</name>
                            <value>iterator?list=[Orange]</value>
                          </element>
                          <element>
                            <name>rti.prototyper.member:x</name>
                            <value>linear?begin=0,end=250</value>
                          </element>
                          <element>
                            <name>rti.prototyper.member:y</name>
                            <value>linear?begin=150,end=200</value>
                          </element>
                          <element>
                            <name>rti.prototyper.member:shapesize</name>
                            <value>linear?begin=30,end=30</value>
                          </element>
```

```
                   </value>
                 </property>
               </datawriter_qos>
           </data_writer>
       </publisher>
   </domain_participant>

   <domain_participant name="ControlledShapeSubscriber"
                domain_ref="MyDomainLibrary::ShapeDomain">

       <subscriber name="MySubscriber">
         <data_reader name="MySquareRdr"   topic_ref="Square"/>
         <data_reader name="MyCircleRdr"   topic_ref="Circle"/>
        <data_reader name="MyTriangleRdr" topic_ref="Triangle"/>
    </subscriber>
</domain_participant>

<domain_participant name="ControlledShapePubSub"
                domain_ref="MyDomainLibrary::ShapeDomain">

    <publisher name="MyPublisher">
      <data_writer name="MyTriangleWr" topic_ref="Triangle">
        <datawriter_qos>
          <property>
            <value>
              <element>
                <name>rti.prototyper.member:color</name>
                <value>iterator?list=[Green,Yellow]</value>
              </element>
              <element>
                <name>rti.prototyper.member:x</name>
                <value>linear?begin=100,end=200</value>
              </element>
              <element>
                <name>rti.prototyper.member:y</name>
                <value>linear?begin=0,end=250</value>
              </element>
              <element>
                <name>rti.prototyper.member:shapesize</name>
                <value>linear?begin=30,end=30</value>
               </element>
            </value>
          </property>
        </datawriter_qos>
      </data_writer>
```

```
104.             </publisher>
105.
106.           <subscriber name="MySubscriber">
107.               <data_reader name="MyCircleRdr" topic_ref="Circle"/>
108.           </subscriber>
109.         </domain_participant>
110.
111.         </participant_library>
112.         </dds>
```

The first thing we notice is that there are no **<types>** or **<domain_library>** sections in this file. This is because the configurations defined in this file are reusing the same data-types and DDS *domains* already defined in the "USER_QOS_PROFILES.xml" so there is no need to define new ones here. It often makes sense to extract all the type and domain information into a separate XML file that is shared, so that configuration files defining specific application scenarios in that DDS *domain* can all reuse the same DDS type and Topic model.

The second thing we observe in **ShapeDemoConfig.xml** is the use of the *property* QoS in the *DataWriter* configuration to specify the values that *Prototyper* will use when publishing data on that *DataWriter*. See lines 17-36, 41-60, and 82-101 within the <properties> tag.

DataWriter properties whose names have the prefix "**rti.prototyper.member**:" are interpreted by *Prototyper* as instructions for setting the value of the data-member whose name follows the ":" character.

For example, the property on line 86, **rti.prototyper.member:color** instructs the *DataWriter* on how to set the color of the Triangle Topics that it publishes. The instructions **"iterator?list=[Green,Yellow]"** tells the *DataWriter* to publish both green and yellow triangles.

An instruction that begins with "linear" specifies a range of values that can be used to set the member. For example, on lines 89- 90, we see:

```
<name>rti.prototyper.member:x</name>
<value>linear?begin=100,end=200</value>
```

This means the value for **x** should be set linearly within the range 100-200. This is consistent with what we see for the "Green" and "Yellow" triangles.

Please see Section 4.3 for more details on the behavior of *Prototyper* and the syntax for constraining the values it publishes.

3 Using Prototyper

Prototyper is a command-line tool. You can control its behavior via command-line options. You can invoke *Prototyper* with the "-help" option to see a list of the valid options and a short summary of each:

On a Windows system:

```
..\..\..\scripts\rtiddsprototyper -help
```

On a UNIX-based system:

```
../../../scripts/rtiddsprototyper -help
```

The command-line options are summarized in the table below.

Option	**Values**	**Purpose**
-help	N/A	Prints a summary of the options available. Example: `-help`
-version	N/A	Print the version of the RTI Connext libraries used. Example: `-version`
-appId	<integer>	Set the applicationId used by the *DomainParticipant* that *Prototyper* creates. Example: `-appId 0x12345678`
-verbosity	<integer>	Sets the verbosity level. Example: `-verbosity 2`
-cfgFile	<string>	Path to an XML file that *Prototyper* should parse to look for participant configurations. **Example:** `-cfgFile ShapeDemoConfig.xml`
-cfgName	<string>	Name of the participant configuration describing the *DomainParticipant* that will be created by *Prototyper*. The configuration name must correspond to one of the participant configurations in the XML files loaded by *Prototyper*. **Example:** `-cfgName ParticipantLibrary::ShapePublisher`
-participantName	<string>	Name used for the participant. It will be propagated in the *DomainParticipant* QoS within the ENTITY_NAME QoS Policy. **Example:** `-participantName MyShapePrototype`
-sendPeriod	<float>	Indicates the period in seconds at which data is sent. Each period, one sample will be written on each *DataWriter* within the *DomainParticipant*. **Example:** `-sendPeriod 1.5`
-runDuration	<float>	Indicates the total time in seconds that *Prototyper* will run. After this time elapsed *Prototyper* will exit. **Example:** `-runDuration 100`
-disableDataFill	N/A	Does not set the values of the data written. In this situation, the values written are the ones that correspond to the data as initialized by the corresponding TypeSupport factory. Typically this sets all scalar values to zero, sequences to zero length, and strings to empty. **Example:** `-disableDataFill`

4 Understanding Prototyper

Prototyper is an emulation tool whose purpose is to facilitate the rapid development and scenario testing of Connext applications. Using *Prototyper,* an application developer or system integrator can quickly answer questions related to the performance of applications such as:

- How big will applications be if they create a certain number of *DataWriters* and *DataReaders* publishing and subscribing to certain *Topics* with specified data-types?
- How much CPU will a specific application consume on a particular hardware platform when publishing data to a set of subscribers under a concrete scenario?
- How long it would take for discovery to occur given a set of computers which a publishing and subscribing certain *Topics*?

- How are some of these answers affected by a change of QoS settings?

Prototyper is an *emulation* rather than a *simulation* tool because it takes the approach of facilitating execution of application scenarios on the real computer and network hardware, as opposed to simulating the results using a computational algorithm and model. For complex distributed applications, the emulation approach is preferred given the many factors that affect ultimate application performance: computer hardware, operating system, network infrastructure, interactions with other applications and processes running on the same systems, etc.

4.1 Workflow

Internally, *Prototyper* executes the workflow shown in the figure below:

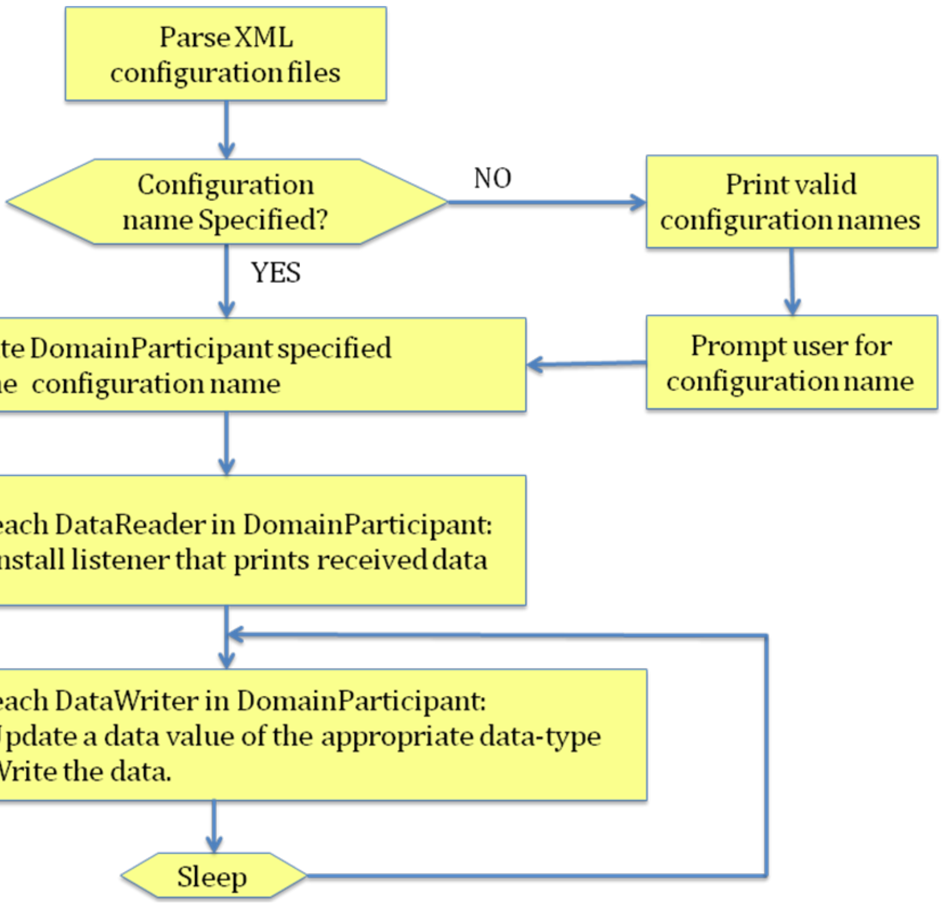

Parse XML configuration files
Configuration name Specified?
NO
Print valid configuration names
YES
Create DomainParticipant specified by the configuration name
Prompt user for configuration name
For each DataReader in DomainParticipant:
1. Install listener that prints received data
For each DataWriter in DomainParticipant:
1. Update a data value of the appropriate data-type
2. Write the data.
Sleep

Figure 3: Workflow of RTI Prototyper

The most important aspects of this workflow are:

- *Prototyper* creates a single *DomainParticipant* identified by its configuration name. All DDS Entities within the *DomainParticipant* are automatically created.
- *Prototyper* installs listeners on all the *DataReader* entities within the *DomainParticipant*. These listeners just print the received data. Whenever an application publishes data in one of the related *Topics*, *Prototyper* receives and prints the data.
- *Prototyper* creates data objects for each *DataWriter* within the *DomainParticipant*.

- *Prototyper* periodically writes data using each *DataWriter* within the *DomainParticipant*. The content of the data is modified each time the data is written. By default, *Prototyper* uses an internal algorithm which cycles through a range of values consistent with the data type. This behavior can be changed, as described in Section 4.3.3.

4.2 Configuration Files Parsed by Prototyper

By default, *Prototyper* looks in the standard location for XML QoS Profile files and loads them if they are found. (See the chapter on *Configuring QoS with XML* in the *RTI Core Libraries and Utilities User's Manual*). The XML QoS Profile files can contain participant configurations as well.

These locations are:

- **$NDDSHOME/resource/qos_profiles_4.5f/xml/NDDS_QOS_PROFILES.xml:**

  This file contains the *RTI Connext* DDS default QoS values; it is loaded automatically if it exists. When present this is the first file loaded.
- **File specified in NDDS_QOS_PROFILES Environment Variable:**

  The files (or XML strings) separated by semicolons referenced in this environment variable, if any, are loaded automatically. These files are loaded after the **NDDS_QOS_PROFILES.xml** and they are loaded in the order they appear listed in the environment variable.
- **<*working* directory>/USER_QOS_PROFILES.xml:**

  This file is loaded automatically if it exists in the 'working directory' of the application, that is, the directory from which the application is run. This file is loaded last.

In addition, *Prototyper* will load the XML file specified by the command-line option **-cfgFile** (see Section 3).

Prototyper will look for participant configurations in all these files. *Prototyper* will exit, printing an error message if no participant configurations are found, or if the participant configuration specified by the command-line option, **-cfgName**, is not found within the loaded files.

4.3 Data Values Written by Prototyper

The value of the data written is changed for each sample written. Unless otherwise specified, *Prototyper* uses a default data-generation algorithm to set each member in the data. You can change this behavior by using property settings in the corresponding *DataWriter*.

The default data-generation algorithm operates differently for non-key data members (also known as regular data members) and for key data members.

4.3.1 Values Set by Default Data-Generation Algorithm on Non-Key Members

The default data-generation algorithm sets every non-key data member in the sample to a value that approximates an incrementing counter, coerced to be appropriate for the member data-type. This approach makes it easy to observe the data and see how progress is being made.

For example, as we saw in the HelloWorld output in section 2.1.1, the default algorithm will set each integer member to the values 0, 1, 2, 3, … in sequence. Float members will be set to the values 0.0, 1.0, 2.0. String members will be set to "String: 0", "String: 1", "String: 2".

The table below describes how the default algorithm sets each regular (non-key) member.

Member Type	**Values Set by Default Data-Generation Function**	**Example**
Integer types: Octet, short, unsigned short, long, unsigned long, long long, unsigned long long	Integer values starting from 0 and incrementing by 1 each sample written.	Member: long x; x will be set to: 0, 1, 2, 3, 4, …
Floating point types: float, double	Floating point values starting with 0.0 and incremented by 1.0 each sample written.	Member: float x; x will be set to: 0.0, 1.0, 2.0, 3.0, 4.0, …
String types: string, wstring	String that follows the pattern: "String: %d" where %d is replaced by the value of a counter that starts at 0 and increments by 1 for each sample written.	Member: string x; x will be set to: "String: 0", "String: 1", "String: 2", "String: 3", "String: 4", …

Enumerated type	Each value of the enumeration, starting with the first and cycling back to the beginning when all the values have been generated.	enum EnumType {
		    A, B, C
		};
		
		Member:
		EnumType x;
		x will be set to:
		A, B, C, A, B, …

Union type	Union discriminator set to always select the last member.
	The value of the last member is set according to its type using the default data-generation function.
		union UnionType (long) {
		    case 1: long along;
		    case 2: float afloat;
		    case 3: string aString;
		};
		
		Member:
		UnionType x;
		
		x will always be set as case 3, with x.aString taking these values:
		"String: 0", "String: 1", "String: 2", "String: 3", …

Array type	Set all elements to the same value using the default data-generation function for the element.	Member:
		long x[3];
		
		x[0], x[1], x[2] will be set to the same incrementing values, e.g.:
		x[0] = x[1] = x[2] = 0
		x[0] = x[1] = x[2] = 1
		x[0] = x[1] = x[2] = 2

Member Type	Values Set by Default Data-Generation Function	Example

Sequence type	Sets the length to 1 and the single element to a value using the default data-generation function for non-key members of that type.	Member:
		sequence<long,10> x;
		Set length to 2 and
		x[0]=x[1]=0,1,2,3,...

4.3.2 Values Set by the Default Data-Generation Algorithm on Key Members

The default data-generation algorithm sets the key data members to a constant value so all samples from a single *DataWriter* correspond to the same data-object instance (same value of the key). The actual value is random but set appropriately for each data type. The table below shows the values set for key members for each member type.

Member Type	Values Set by Default Data-Generation Function	Example
Integer types: octet short, unsigned short, long, unsigned long, long long, unsigned long long	Random integer value.	Member: long x; //@key x will be set to a random value, e.g.: 8761237
Floating point types: float, double	Random floating value.	Member: float x; //@key x will be set to a random value, e.g.: 3.741723
String types: string, Wsting	String that follows pattern: “Key: RR” where RR is a random number.	Member: string x; //@key x will be set to a random string, e.g.: “Key: 76896713”

Enumerated type	Random member of the enumeration.	enum EnumType {
		A, B, C
		};
		
		Member:
		EnumType x;
		
		x will be set to a random element, e.g.: B

Member Type	**Values Set by Default Data-Generation Function**	**Example**

Union type	Union discriminator set to always select the last member.
	The value of the last member is set according to its type using the default data-generation function for a key member.
		union UnionType (long) {
		case 1: long along;
		case 2: float afloat;
		case 3: string aString;
		};
		
		Member:
		UnionType x;
		
		x will be set as case 3 and x.aString set to a random string, e.g.: "Key: 76896713"

Array type	Sets all elements to the same value using the default data-generation function for key member of that type.	Member: long x[3]; x[0], x[1], x[2] will be set to the same random value, e.g.: 8761237
Sequence type	Sets the length to 1 and the single element to a value using the default data-generation function for key member of that type.	Member: sequence<long,10> x; Sets length to 1 and x[0] to a random value, e.g.: 8761237

4.3.3 Controlling the Data Values Written by Prototyper

The application can change the default data-generation algorithm so that the values are more appropriate for the scenario being prototyped. Currently this feature is quite limited. It will be expanded in the future.

The value of the data-members published by *Prototyper* can be controlled by setting the Property QoS Policy for the corresponding *DataWriter*.

The Property QoS is a general QoS policy in the *DataWriter*. It accepts a sequence of property (name, value) pairs and can be used for many purposes. This QoS policy is described in detail in the *Core Libraries and Utilities User's Manual* (section 6.5.17).

Prototyper looks at the properties in the *DataWriter* whose names have the prefix "rti.protoptyper" and uses corresponding value fields to control the behavior of that member as it relates to the *DataWriter*.

To change the values that *Prototyper* writes for a specific *DataWriter* you set the property with a name that follows the pattern:

```
rti.prototyper.member:<replace with the name of the member>
```

The property is set to a value that describes the data-generation function for that member. This property value follows the pattern:

```
<function name>?<parameter 1>=<value1>,<parameter2>=<value2>,…
```

For example, we used the following property in the example in Section 2.2.4 to specify that *Prototyper* should set the 'color' member to the values "Green" and "Yellow", successively.

```
<element>
    <name>rti.prototyper.member:color</name>
    <value>iterator?list=[Green,Yellow]</value>
</element>
```

Recall that the corresponding data-type for the *topic* written by the *DataWriter* was defined as:

```
           <struct name="ShapeType">
               <member name="color"  key="true"
                       type="string"  stringMaxLength="MAX_COLOR_LEN"/>
               <member name="x" type="long"/>
               <member name="y" type="long"/>
               <member name="shapesize" type="long"/>
           </struct>
```

So "color" corresponds to the name of a member of type string.

To specify how to set the values of a nested member, provide the full name of the member using the '.' character to navigate to the nested sub-structures, just as you would do if you were to access the member from a language such as C/C++ or Java.

For example, assume the following data-type schemas for Plane and Coordinates.

```
<types>
     <struct name="Coordinates">
         <member name="latitude" type="long"/>
         <member name="longitude" type="long"/>
         <member name="height" type="long"/>
     </struct>

     <struct name="Plane">
         <member name="airline" type="string" key="true"/>
         <member name="flight_num" type="long" key="true"/>
         <member name="coordinates"  type="nonBasic"
                 nonBasicTypeName="Coordinates"/>
      </struct>
</types>
```

To specify that *Prototyper* should set the values of the latitude within the coordinates of a Plane linearly between 20 and 60, set the following property on the *DataWriter* that is publishing the Plane data:

```
<element>
   <name>rti.prototyper.member:coordinates.latitude</name>
   <value>linear?begin=20,end=60</value>
```

```
</element>
```

The values of array or sequence members can also be specified:

- To specify how to set *all* the elements in the array or sequence, use the name of the array or sequence field followed by empty square brackets ([]).
- To specify how to set *a specific* member at index **k**, use the name of the array or sequence field followed by a "[k]" string. (where **k** should be replaced with the actual value of the index you want to control).

Sequences support having a length that is smaller than their capacity (maximum length). To facilitate control of sequence length, the following heuristic is used: If a rule specifies how an element of the sequence at a particular index "k" should be set, then the length of the sequence will be adjusted to be at least "k+1" such that the sequence can contain an element at index "k". There are two exceptions to this rule: (1) if the index exceeds the capacity, and (2) if an explicit rule has been specified for the length of the sequence itself. Setting the length of the sequence explicitly is described in the next few paragraphs.

For example, assume the following data-type schemas for TruckFleet, Truck, and Location.

```
<types>
    <const name="MAX_TRUCKS" type="long" value="1024"/>

    <struct name="Location">
        <member name="latitude" type="long"/>
        <member name="longitude" type="long"/>
    </struct>

    <struct name="Truck">
        <member name="license_plate" type="string" key="true"/>
        <member name="location"  type="nonBasic"
                nonBasicTypeName="Location"/>
    </struct>

    <struct name="TruckFleet">
        <member name="fleet_name" type="string" key="true"/>
        <member name="trucks"  type="nonBasic"  nonBasicTypeName="Truck"
                sequenceMaxLength="MAX_TRUCKS"/>
    </struct>
</types>
```

To specify that all Trucks must have a location with latitude set linearly between 40 and 65:

```
<properties>
    <value>
      <element>
        <name>rti.prototyper.member:trucks[].location.latitude</name>
        <value>linear?begin=40,end=65</value>
      </element>
    </value>
</properties>
```

Since there are no rules that specify the sequence length or the elements at a specific index, the length of the sequence will be set to 1.

The following properties show how to specify that the first Truck (the one at index 0) in the fleet should have its license plate set to “CA91RTI6” and its latitude Location set linearly to values between 40 and 45. They also specify that the fourth truck (the one at index 3) should have its license plate set to “CA94NDDS7” and its latitude Location set linearly to values between 60 and 65:

```xml
<properties>
    <value>
      <element>
        <name>rti.prototyper.member:trucks[0].license_plate</name>
        <value>iterator?list=[CA91RTI6]</value>
      </element>
      <element>
        <name>rti.prototyper.member:trucks[0].location.latitude</name>
        <value>linear?begin=40,end=45</value>
      </element>
      <element>
        <name>rti.prototyper.member:trucks[3].license_plate</name>
        <value>iterator?list=[CA94NDDS7]</value>
      </element>
      <element>
        <name>rti.prototyper.member:trucks[3].location.latitude</name>
        <value>linear?begin=60,end=65</value>
      </element>
    </value>
</properties>
```

Since there are rules specifying how to set elements 0 and 3 of the sequence, the sequence length will be set to 4 (such that it can have elements with indices 0, 1, 2, and 3).

The lengths of sequences can also be controlled explicitly. This is done by using the name of the array followed by the suffix ‘#length’. For example to specify that *Prototyper* should write samples that contain from 6 to 10 Trucks, linearly you can use the following properties:

```xml
<properties>
    <value>
      <element>
        <name>rti.prototyper.member:trucks#length</name>
        <value>linear?begin=6,end=10</value>
      </element>
    </value>
</properties>
```

If the length of the sequence is specified explicitly as above, then it takes precedence over specifications of the values of sequence elements at particular indices. This means that first the length of the sequence is determined according to the explicit rule, and then any rules that apply to the elements with indices between 0 and length-1 are applied. Rules for elements with indices outside the length are ignored.

For example, the following properties specify that *Prototyper* should write samples that contain from 6 to 10 Trucks, linearly, and that any Truck with an index of 8 will be assigned the license place "CA8888".

```
<properties>
    <value>
      <element>
        <name>rti.prototyper.member:trucks#length </name>
        <value> linear?begin=6,end=10</value>
      </element>
      <element>
        <name>rti.prototyper.member:trucks[8].license_plate</name>
        <value>iterator?list=[CA8888]</value>
      </element>
    </value>
</properties>
```

In the example above, successive "TruckFleet" samples will set the 'trucks' sequence to a length of 7, 8, 9, and 10. Samples with lengths 9 and 10 will set the *license_plate* member of the Truck at index 8 to "CA 8888". Samples with 6, 7, and 8 Trucks will not set that member because its index exceeds what the *trucks* array can accommodate.

The choices available for the data-generation algorithms are listed in the table below. As mentioned, this feature is currently limited. It will be extended in future releases.

Generation Function	**Parameters**	**Example**	**Description**
linear	begin, end, numsteps	linear?begin=0,end=100,numsteps=50	Generates 'numsteps' linearly-spaced values between and including 'begin' and 'end'.
iterator	list	iterator?list=[RED,GREEN,BLUE]	Set each element in the list in sequence
Init	N/A	Init	Set the value to the initialization value for the type. This is typically zero for atomic types, zero length sequences, etc.