RTI Recording Service

User's Manual

Version 5.0

Your systems. Working as one.

Trademarks

Copy and Use Restrictions

Third-Party Copyright Notices

Technical Support

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

Contents

8 Accessing the Replay Tool from a Remote Location

9 Viewing Recorded Data with SQLite

10 Exporting Recorded Data

11 Example Configuration Files

Chapter 1 Welcome to RTI Recording Service

RTI® *Recording Service* includes:

- ❑ ***Recording Console***, a simple graphical user interface (GUI) for using the *Record* and *Replay* tools. This interface significantly reduces *Recording Service* configuration time and complexity, and does not require any programming. The *Recording Console* makes it easy to use *Recording Service* for testing algorithms and other processing logic against pre-recorded test data, conducting regression testing from 'golden' data inputs, or recording live data from the field for post-mission analysis. See Chapter 2: Using Recording Console.
- ❑ ***Record***, an *RTI Connext*™ (formerly *RTI Data Distribution Service*) application that records both *RTI Connext* discovery and topic data. All recorded data is stored in one or more SQL database files. See Chapter 3: Using the Record Tool.
- ❑ ***Replay***, a tool that can ‘play back’ the recorded data. You even have the option of replaying the data with different data rates or QoS settings. See Chapter 6: Using the Replay Tool.
- ❑ ***Convert***, a utility that enables serialized or deserialized data recorded with *Record* to be exported to CSV, HTML, SQL, or XML formats (see Chapter 9: Viewing Recorded Data with SQLite).
- ❑ A SQL command-line tool, **sqlite3**, which provides another way to view the data files.

Recording Features

- ❑ Records data from applications in multiple domains.
- ❑ Records entire Topics, or specific Topic fields, based on POSIX file-name matching expressions.
- ❑ Records all data types except bit-fields.

- ❑ Records to multiple files with configurable file-size limits. Optionally overwrites the oldest file when the maximum number of files has been reached.
- ❑ Records the DDS SampleInfo structure and a timestamp for both discovery data and user data.
- ❑ Records using either Best Effort or Reliable communications.
- ❑ Optionally records data from only specified partitions.
- ❑ Supports remote operation.

Replay Features

- ❑ Publishes data samples that were recorded in serialized format.
- ❑ Highly configurable—you can:
  - Choose which serialized topics to replay
  - Set the replay rate (faster or slower) or use the original rate
  - Change the QoS of the publications
  - Configure the QoS for the tool itself
  - Dynamically control the replay (start, stop, pause) and single-step through the data samples

This document assumes you have a basic understanding of DDS terms such as *DomainParticipants, Publishers, DataWriters, Topics,* and Quality of Service (QoS) policies. For an overview of DDS terms, please see the *RTI Core Libraries and Utilities User's Manual.*

Chapter 2 Using Recording Console

This chapter describes how to use *Recording Console*, which provides an easy way to record and replay data.

2.1 Starting and Stopping the Console

Recording Console's executable is in **<*install directory*>/console**. A script to run the executable is in **<*install directory*>/scripts**.

On Linux systems:

1. Open a command prompt and change to the **<*install directory*>/scripts** directory
2. Start the *Console* by entering:
   > **rtirecordingconsole**
3. Set a Domain ID as described in Section 2.2.

On Windows systems:

1. From the **Start** menu, navigate to **RTI Recording Service <*version*>**, and select **Recording Console**.
2. Set a Domain ID as described in Section 2.2.

Figure 2.1 **Console at Startup**

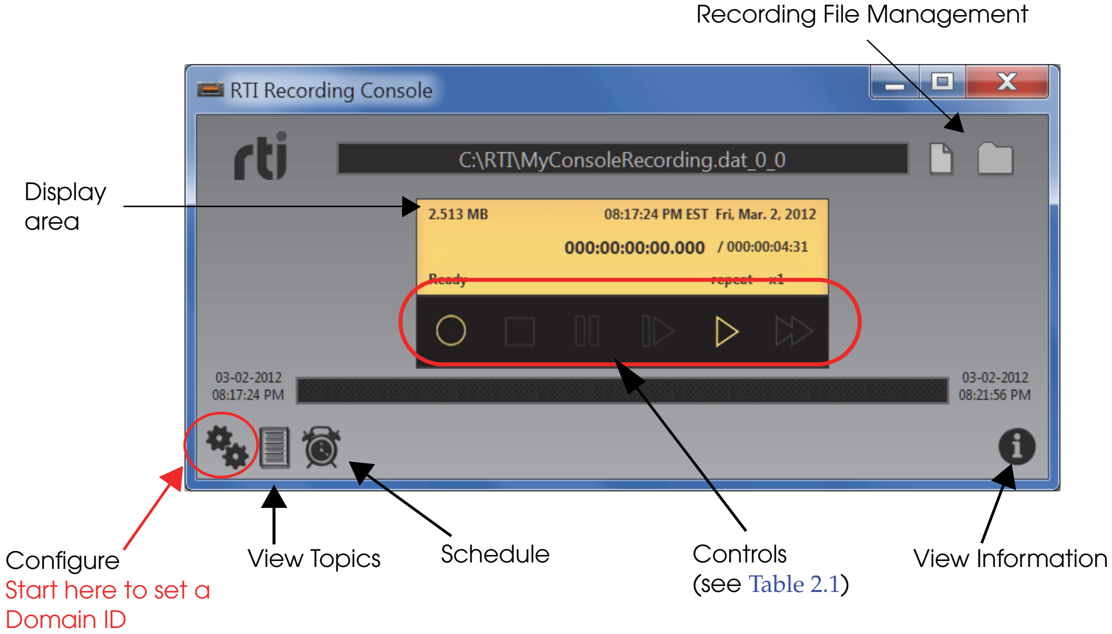


Table 2.1 **Console's Controls**

○	□	‖	|▷	▷	▷▷
Record	Stop	Pause	Single Step (Press Pause first)	Play	Fast Forward

2.2 Configuring Recording Console

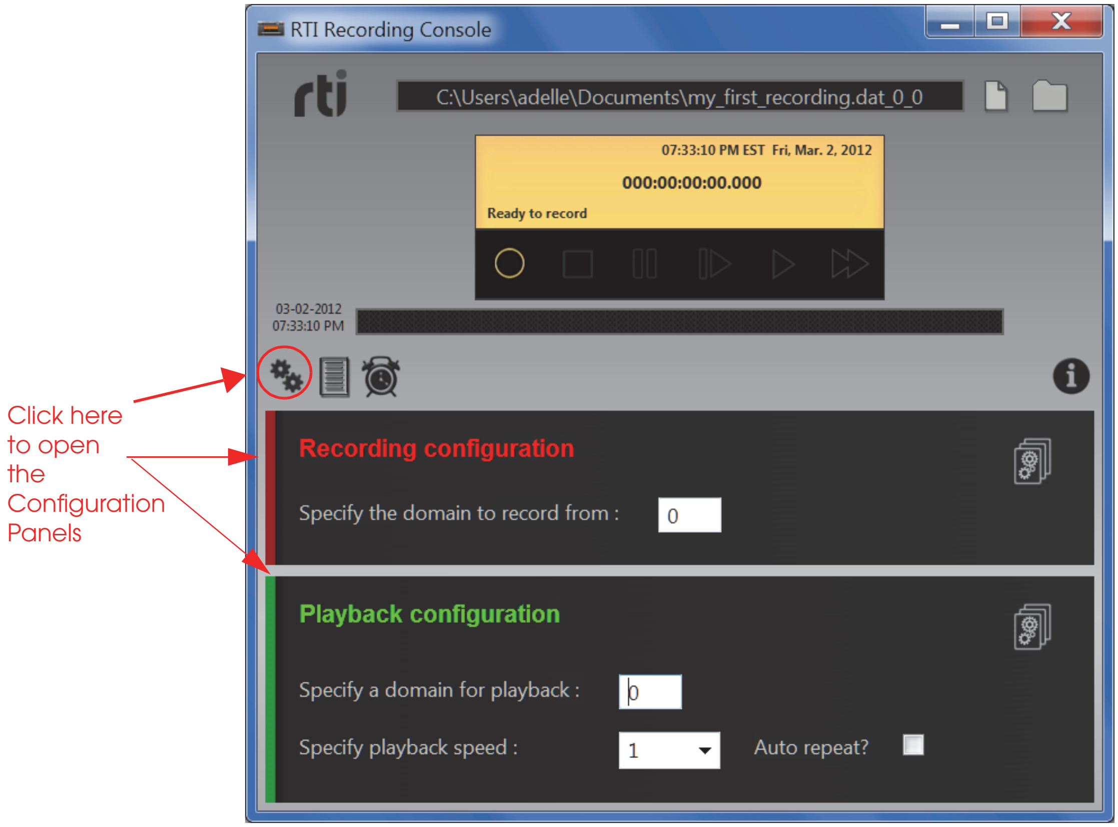
Before you can use *Recording Console* to record or replay data, you must specify a domain ID in the Configuration panels.

- ❏ To record data, specify the domain in which the data you want to record is being published.
- ❏ To replay data, specify the domain that you want to publish the previously recorded data into.

By default, *Recording Console* records and replays data using the default *DomainParticipant* QoS settings described in the *Connext* documentation. If your *Connext* applications use default *DomainParticipant* QoS settings (including transport settings), you can record and replay data 'out of the box'—with no QoS changes.

However, if your system uses any *DomainParticipant* QoS settings that would be incompatible with the default settings, you need to write a configuration file that can be loaded by *Recording Console* to use when recording or replaying the data.

There are two ways to configure *Recording Console*: by using its Configuration panels, shown below, or by using settings from external configurations files (see Section 2.2.1).



Click here to open the Configuration Panels

C:\Users\adelle\Documents\my_first_recording.dat_0_0

Ready to record

Recording configuration

Specify the domain to record from : 0

Playback configuration

Specify a domain for playback : 0

Specify playback speed : 1

Auto repeat? □

2.2.1 Configuring from an External File

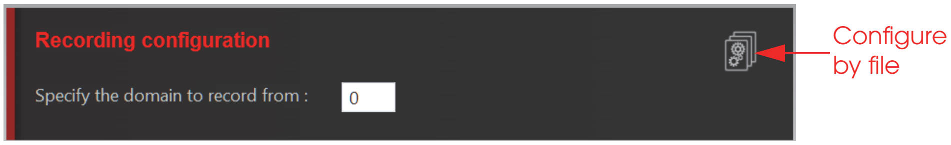
If you have a use case which is not covered by the default configuration generated by *Recording Console*, you can use an external configuration file as the basis of the settings to record or replay.

Recording Console can load any configuration file which is supported by the *Record* or *Replay* tools. These files are described in Chapter 4: Configuring the Record Tool and Chapter 7: Configuring the Replay Tool.

2.2.1.1 Configuring Recording from an External File

To use an external configuration file for recording:

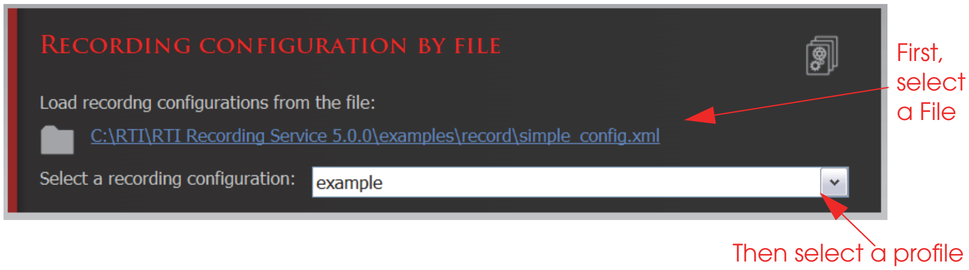
1. Press to open the Recording and Playback Configuration panels.
2. In the Recording Configuration panel, press to open the Configuration by File panels.



Recording configuration

Specify the domain to record from : 0

3. Press the Open Folder button to select a configuration file for recording.



RECORDING CONFIGURATION BY FILE

Load recordng configurations from the file:

C:\RTI\RTI Recording Service 5.0.0\examples\record\simple_config.xml

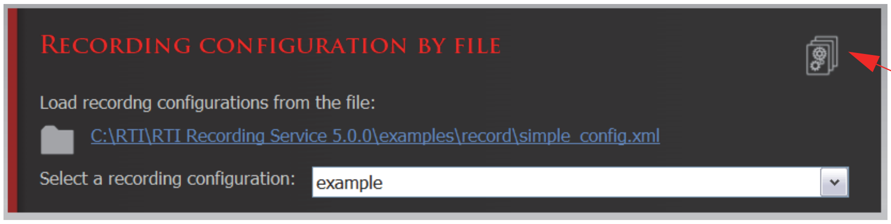
Select a recording configuration: example

4. Select a QoS profile from the drop-down listbox.

It is important to understand which parts of the configuration settings are used from the external file. These configuration elements from the file are retained (will be used):

- ❑ <dds><recorder>: The name attribute will be used for the launched service.
- ❑ <dds><recorder><remote_access><remote_access_domain>: This will be used for the administration domain ID.
- ❑ <dds><recorder>: Many of the settings from this element are retained, except as noted below:
    - <remote_access>: The Domain ID is retained. All other settings are replaced so that they are compatible with *Recording Console*'s settings.
    - <recorder_database>: This is replaced with the database file specified in *Recording Console*.

Note: To return to using the default QoS settings, press ⚙ (on the far right).



RECORDING CONFIGURATION BY FILE

Load recordng configurations from the file:

C:\RTI\RTI Recording Service 5.0.0\examples\record\simple_config.xml

Select a recording configuration: example

To return to default configuration

2.2.1.2 Using Domain 99 in External File Configurations

By default, *Recording Console* monitors and controls the Record and Replay services on domain 99.

When configuring either the Record or Replay services from a file, if you happen to be using domain 99 as your recording or playback domain, we recommend that you change the administration domain from 99 to a different domain ID, by either specifying it in the configuration file or by editing the file, **settings.ini**, which should be located in your home directory:

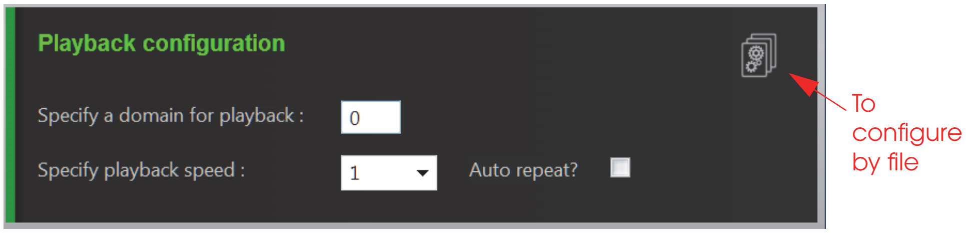
- ❑ On Windows Systems:
  My Documents\RTI\RTI Recording Service 5.0.x\console\settings.ini
- ❑ On Linux systems:
  ~/RTI/RTI Recording Service 5.0.x/console/settings.ini

Again, this recommendation only applies if the domain you are using in your file configuration is domain 99.

2.2.1.3 Configuring Replay from an External File

To use an external configuration file for replay:

1. Press to open the Recording and Playback Configuration panels.
2. In the Playback Configuration panel, press to open the Configuration by File panels.



Playback configuration

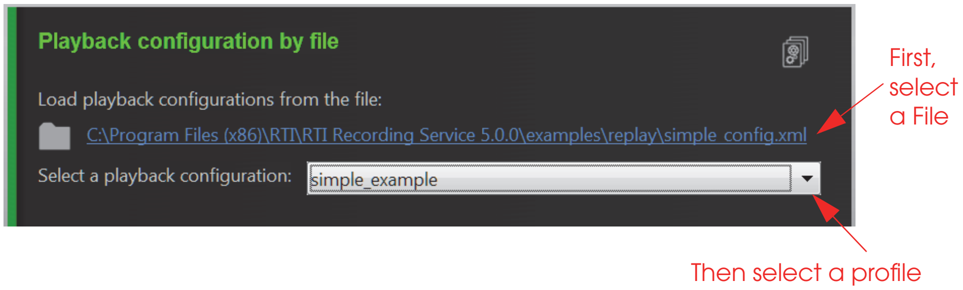
Specify a domain for playback : 0

Specify playback speed : 1

Auto repeat? □

To configure by file

3. Press the Open Folder button to select a configuration file for replay.



Playback configuration by file

Load playback configurations from the file:

C:\Program Files (x86)\RTI\RTI Recording Service 5.0.0\examples\replay\simple_config.xml

Select a playback configuration: simple_example

First, select a File

Then select a profile

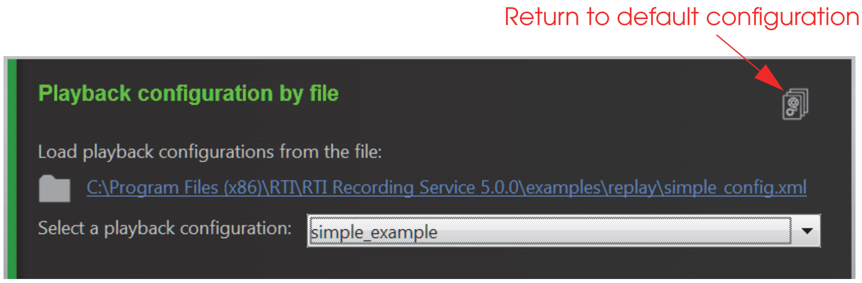
4. Select a QoS profile from the drop-down listbox.

Here again, only parts of the configuration file are retained, while other settings are taken from *Recording Console*. These configuration elements are retained:

- ❑ <dds><replay_service>: The name attribute will be used for the launched service.
- ❑ <dds><replay_service><administration><domain_id>: This will be used for the administration domain ID. The rest of the <administration> element is replaced to ensure run-time compatibility with *Recording Console*.
- ❑ <dds><replay_service>: Many of the settings will be retained from this element, except for:
  - Only the first <session> from the first <replay_database> will be retained.

- The <filename> will be replaced with the file specified in *Recording Console*.

Note: To return to using the default QoS settings, press (on the far right).



Return to default configuration

Playback configuration by file

Load playback configurations from the file:

C:\Program Files (x86)\RTI\RTI Recording Service 5.0.0\examples\replay\simple_config.xml

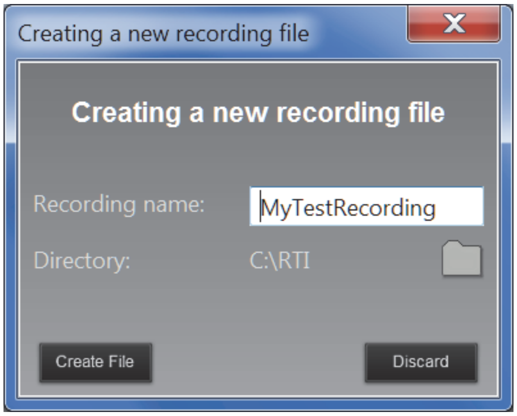
Select a playback configuration: simple_example

2.3 Recording Data

To record data:

1. Make sure you have set the domain ID in the Record Configuration panel (see Section 2.2).
2. Choose where to save the recorded data. You can create a new file or choose an existing one:
   - To create a new file: Press the New Recording button and specify a file name and location for the new recording. Then click on **Create File**.
   - To record over an existing file: Press the Open Folder button in the upper-right corner, locate the file that you want to record into.

Note: If you specify an existing file, the file will be overwritten with new data. New data is *not* appended to the end of the existing file contents.



Creating a new recording file

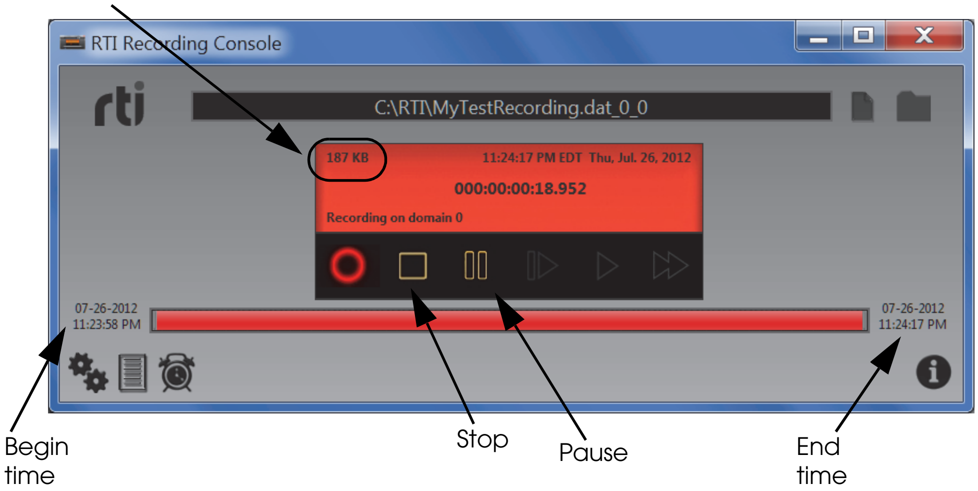
Creating a new recording file

Recording name: MyTestRecording

Directory: C:\RTI

3. Press the Record button ○ to start recording.

File size grows as data is recorded



RTI Recording Console

C:\RTI\MyTestRecording.dat_0_0

187 KB

11:24:17 PM EDT Thu, Jul. 26, 2012

000:00:00:18.952

Recording on domain 0

07-26-2012 11:23:58 PM

07-26-2012 11:24:17 PM

Begin time

Stop

Pause

End time

2.3.1 Troubleshooting Recording Problems

Problem—You pressed the Record button, but the recording file size stays at zero.

Solution—Make sure that:

- ❑ The Recording Domain ID matches the domain ID used by the source (the application from which you want to record data).
- ❑ Data is coming in from the source, by using tools such as *rtiddsspy* (provided with *Connext* in its **/scripts** directory).
- ❑ You have access rights to create files in the directory where the recording file is to be created.

2.4 Replaying Data

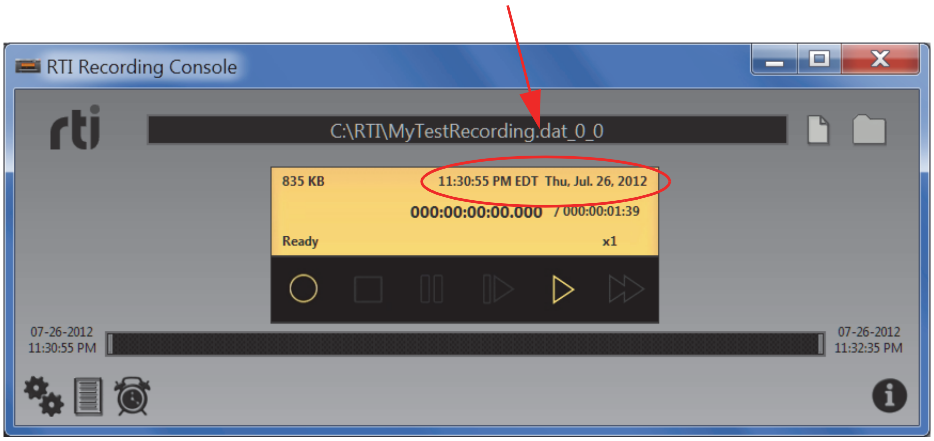
You can use the *Console* to replay data that was recorded using the *Console* or the *Record* tool. You may replay data recorded with an older version of *Recording Service*.

To replay data:

1. Make sure you have set the domain ID in the Playback Configuration panel (see Section 2.2).
2. Press the Open Folder button 📁 in the upper-right corner, locate the file whose data is to be replayed, then click **Open**.

When the file is loaded, you will see the time of the original recording:

Original recording time and date



RTI Recording Console

C:\RTI\MyTestRecording.dat_0_0

835 KB

11:30:55 PM EDT Thu, Jul. 26, 2012

000:00:00:00.000 / 000:00:01:39

Ready

x1

07-26-2012
11:30:55 PM

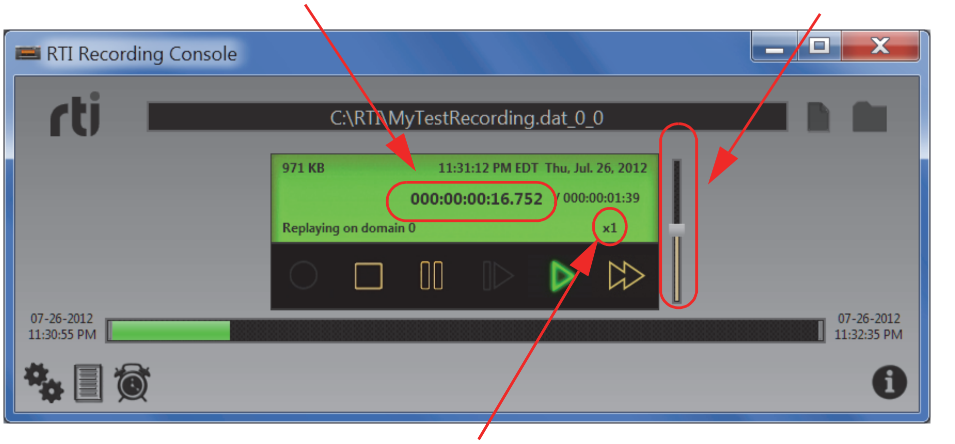
07-26-2012
11:32:35 PM

3. Press Play ▷ to begin replaying the data.

   The display will show you the elapsed time since the start of the replay.

Elapsed time

Slider controls
replay speed



RTI Recording Console

C:\RTI\MyTestRecording.dat_0_0

971 KB

11:31:12 PM EDT Thu, Jul. 26, 2012

000:00:00:16.752 / 000:00:01:39

Replaying on domain 0

x1

07-26-2012
11:30:55 PM

07-26-2012
11:32:35 PM

Replay rate. For example.
x1 = original rate,
x2 = twice the original rate.

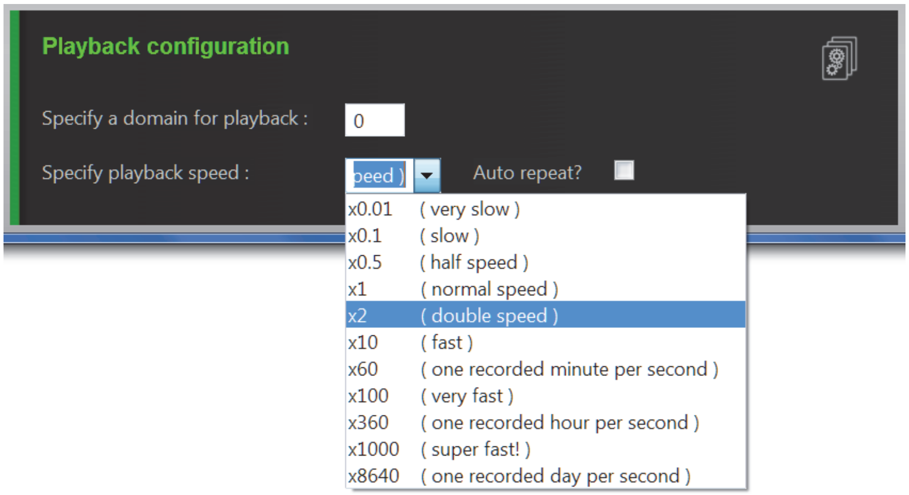
Changing the Replay Rate:

The vertical slider on the right controls the replay speed (up for faster, center for original speed, lower for slower).

To return to the original replay rate, press Play ▷.

You can specify a default replay rate in the Playback configuration panel.

Select a speed from the drop-down list or type in your own value.



Playback configuration

Specify a domain for playback : 0

Specify playback speed : peed)

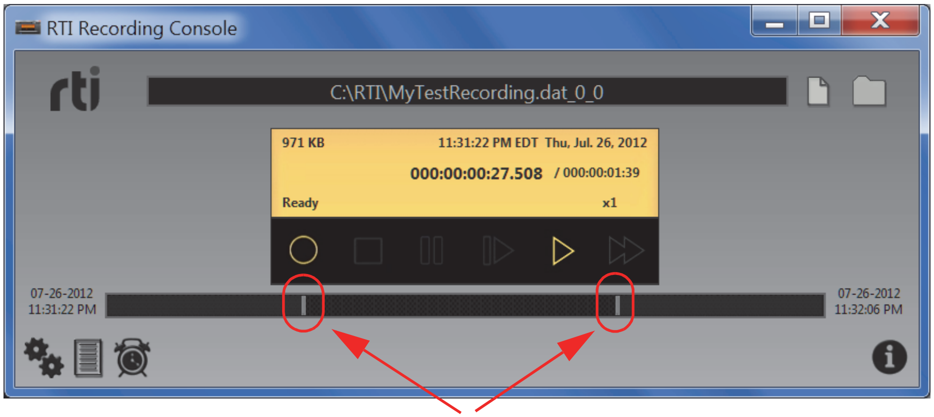
Auto repeat? □

x0.01	(very slow)
x0.1	(slow)
x0.5	(half speed)
x1	(normal speed)
x2	(double speed)
x10	(fast)
x60	(one recorded minute per second)
x100	(very fast)
x360	(one recorded hour per second)
x1000	(super fast!)
x8640	(one recorded day per second)

To automatically repeat the replay in a continuous loop, select the **Auto repeat** checkbox.

Restricting the Time Range to be Replayed

You can limit the start and end time for replaying data by dragging the gray bars seen below:.



Drag these gray bars inward to restrict the time range for replaying data

Note: This feature cannot be used when using a configuration file.

2.5 Viewing Recorded Topics

While recording, or when you have loaded a pre-recorded file, you can use the Recorded Topics panel to see the topics that have been recorded. For each topic, the table shows the topic name, and (when the recording is not in progress) the first and last recorded samples of that topic.

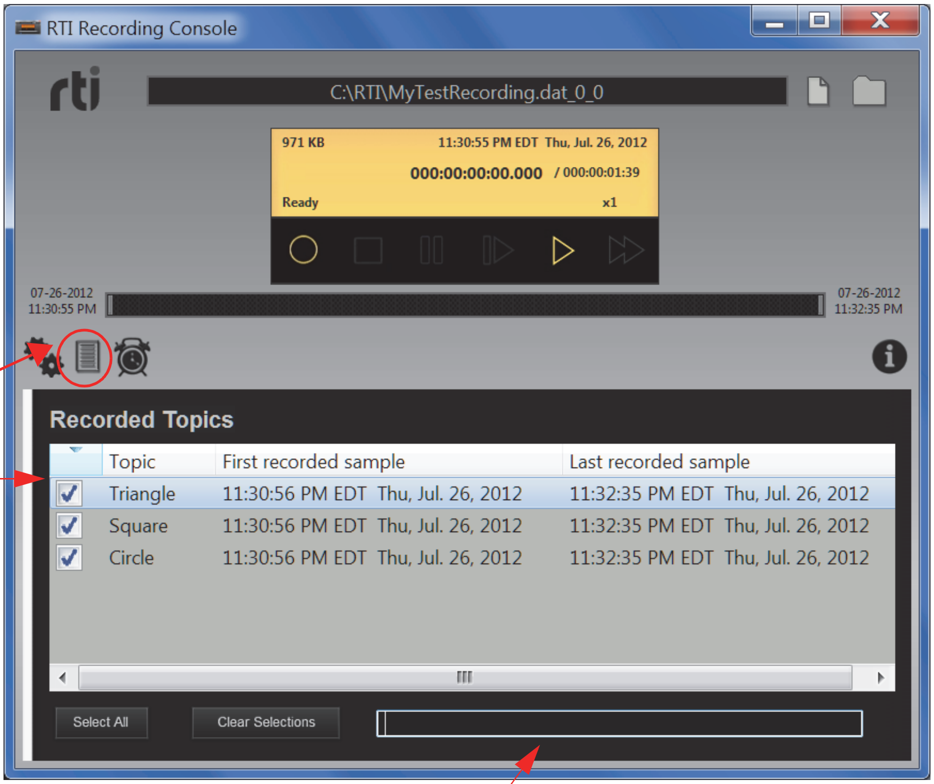
If playback is configured through *Recording Console*, the topic table enables you to select which topics to replay.

Searching for Topics

To assist in selecting multiple topics, use the search bar on the bottom of the topics panel. You can narrow down the topics that are displayed based on a substring in the topic name.

Note: The search bar does not support regular expressions.

When the desired group of topics is displayed, you can use the select all/unselect all buttons on the entire group. Only the selected Topics will be replayed.



	Topic	First recorded sample	Last recorded sample
☑	Triangle	11:30:56 PM EDT Thu, Jul. 26, 2012	11:32:35 PM EDT Thu, Jul. 26, 2012
☑	Square	11:30:56 PM EDT Thu, Jul. 26, 2012	11:32:35 PM EDT Thu, Jul. 26, 2012
☑	Circle	11:30:56 PM EDT Thu, Jul. 26, 2012	11:32:35 PM EDT Thu, Jul. 26, 2012

Click here to open the Recorded Topics Panel

Search for topics by name here

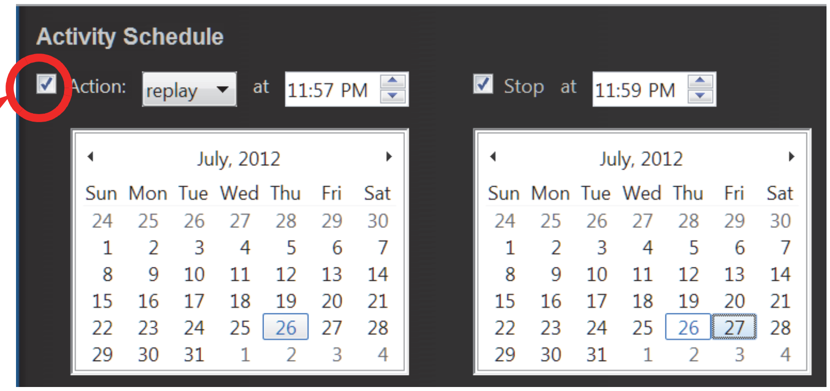
To restore and see the list of all topics, remove (erase) the search string from the search box.

Topic selection in the table is only available when the *Console* is in Ready mode (not recording or replaying data) and you have used the Playback Configuration panel (not a configuration file).

2.6 Scheduling Recording and Replay Tasks

To schedule recording or replay:

1. Press the Schedule button ⏰.
2. Select the type of task (record, replay, or stop current operation) and the starting time and date.
3. Enable the activity by selecting the **Action** checkbox on the left.
4. *Optionally,* select an ending time for the activity.



Must be checked

Activity Schedule

☑ Action: replay at 11:57 PM

☑ Stop at 11:59 PM

Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Important Notes:

- ❑ Be sure to enable the activity by selecting the **Action** checkbox on the left.
- ❑ *Recording Console's* window must remain active for the scheduled operation to run. You may minimize the window, but closing it will cancel the activity.
- ❑ When selecting a file in which to record, be aware that any data already in the file will be erased.

Chapter 3 Using the Record Tool

Besides using *Recording Console*, you can also record data by using the *Record* tool. While the *Console* provides a simple graphical user interface (GUI) for using the *Record* tool, you can also run it directly, without using the *Console*. You may find this method of recording useful when you want to tie its service into your own infrastructure or software or if you need to use its more advanced features. For instance, perhaps you want to run them from your own script to record periodically or to process the recorded data automatically.

See also:

- ❏ Chapter 4: Configuring the Record Tool
- ❏ Chapter 5: Accessing the Record Tool from a Remote Location

3.1 Starting the Record Tool

Open a command prompt[1] and change to the ***<install-dir>*****/scripts** directory. Then enter:

```
> rtirecord -cfgFile <file> -cfgName <configuration>
```

To see a list of available arguments, enter **rtirecord -help**.

To see which configurations are available, use the **-listCfgs** option:

```
$ rtirecord -listCfgs
```

[1] On Windows systems: from the **Start** menu, select **Accessories, Command Prompt**.

To use your own configuration file:

```
$ rtirecord -cfgFile config-file.xml -cfgName my_record_cfg
```

Table 3.1 describes the command-line options and which ones are required.

Chapter 4: Configuring the Record Tool describes the contents of the configuration file. Example files are provided in the ***<installation directory>*/examples** directory.

Table 3.1 **Record Tool's Command-Line Options**

Command-line Option	Description
`-appName <name>`	Assigns an application name to the DomainParticipants created by the *Record* tool. If not specified, the same name used in **-cfgName** will be used.
`-cfgFile <file>`	**Required.** Specifies the XML configuration file (path and filename). In addition to the file provided using this command-line option, the *Record* tool can load other XML files—see Section 4.1.
`-cfgName <name>`	**Required.** This name is used to find the matching **<recorder>** tag in the configuration file.
`-dbName`	Names the fileset to use for storing recorded data. Default: **rtirecord.dat**
`-help`	Prints version information and list of command-line options.
`-licenseFile <file>`	Specifies the license file (path and filename). Only applicable to licensed versions of the *Record* tool. If not specified, the *Record* tool looks for the license as described in Section 2.3 in the *Getting Started Guide*.
`-listCfgs`	Lists the available configuration profiles.
`-verbosity`	Specifies what type of logging information should be printed. 0: silent (Core Libraries and the *Record* tool) 1: errors (Core Libraries and the *Record* tool) (default) 2: warnings (the *Record* tool only) 3: warnings (Core Libraries and the *Record* tool) 4: information (the *Record* tool only) 5: tracing (the *Record* tool only) 6: tracing (Core Libraries and the *Record* tool) This property can also be set in the configuration file. However, this command-line option overrides the value specified in the configuration file.

3.2 Stopping the Record Tool

To stop the Record tool: Press **Ctrl-C**. The *Record* tool will close all files and perform a clean shutdown.

You can also start, stop, and even reconfigure the *Record* tool remotely—see Chapter 5: Accessing the Record Tool from a Remote Location.

Chapter 4 Configuring the Record Tool

When you start the *Record* tool, you may specify a configuration file in XML format (it is not required). In that file, you can set properties that control what to record, how to record, and where to save the recorded data. This chapter describes how to write a configuration file.

4.1 How to Load the XML Configuration

The *Record* tool loads its XML configuration from multiple locations. This section presents the various approaches, listed in load order.

The first three locations only contain QoS Profiles and are inherited from *Connext* (see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*).

❑ **$NDDSHOME/resource/qos_profiles_5.0.x[1]/xml/NDDS_QOS_PROFILES.xml**

This file contains the *Connext* default QoS values; it is loaded automatically if it exists. *(First to be loaded.)*

❑ **File in NDDS_QOS_PROFILES**

The files (or XML strings) separated by semicolons referenced in this environment variable are loaded automatically.

❑ ***<working directory>*/USER_QOS_PROFILES.xml**

This file is loaded automatically if it exists. If the USER_QOS_PROFILES file is found and there is a default profile specified in it, this default profile is automatically applied to the QoS settings of the *Recording Service* entities.

[1] *x* stands for the version letter of the current release.

The next locations are specific to *Recording Service*.

- ***<rtirecord executable location>*/../resource/xml/ RTI_RECORDING_SERVICE.xml**

  This file contains the default configuration for the *Record* tool; it is loaded if it exists. **RTI_RECORDING_SERVICE.xml** defines a configuration that records all topics on domain 0.

- ***<working directory>*/USER_RECORDING_SERVICE.xml**

  This file is loaded automatically if it exists.

- **File specified with the command-line option, -cfgFile** (see Table 3.1 on page 3-2).

- **File specified using the remote command 'configure'**

  The **configure** command (see Table 3.1 on page 3-2) allows loading of an XML file remotely. The file loaded using this command replaces the file loaded using the **-cfgFile** command-line option. *(Last to be loaded.)*

You may use a combination of the above approaches.

4.2 General Format

The configuration file uses XML format. The main sections are:

- General Properties for the Record Tool (Section 4.3)
- Database (Output File) Properties (Section 4.4)—contained in the top-level tag, <record_database>
- Domain Properties (Section 4.5)—contained in the top-level tag, <domain name="String">
- TopicGroup Properties (Section 4.6)—contained in the top-level tag, <topic_group>
- RecordGroup Properties (Section 4.7)—contained in the top-level tag, <record_group>
- Remote Access Properties (Section 4.8)—contained in the top-level tag, <remote_access>
- Domain Type Configuration (Section 4.9)—contained in the top-level tag, <domain_type_config>

Let’s look at a very basic configuration, just to get an idea of its contents. You will learn the meaning of each line as you read the rest of this chapter.

```
<dds>
    <!-- This simple configuration records all topics from domain ID 0 -->

    <recorder name=”example”>
       <!-- Specify where to store the recorded data. -->
       <record_database>
           <database_name> simple_config.dat </database_name>
       </record_database>

       <!-- Create a DomainParticipant in domain 0 with default QoS -->
       <domain name="domain0">
          <domain_id> 0 </domain_id>
          <deserialize_mode>RTIDDS_DESERIALIZEMODE_ALWAYS</deserialize_mode>
       </domain>

       <!-- Create a TopicGroup. A TopicGroup is a collection of Topics
           whose names match the topic_expr. The field_expr specifies
           which fields in the Topics to record. Note that a TopicGroup is
           not bound to a particular domain yet. In this example, the
           TopicGroup All means all fields in all Topics -->

       <topic_group name="All">
           <topics>
              <topic_expr> * </topic_expr>
           </topics>
           <field_expr> * </field_expr>
       </topic_group>

       <!-- Create a RecordGroup. A RecordGroup controls which TopicGroups
        are recorded for a set of domains. Each recorded Topic is stored
        in a table with the format "record_group.domain.Topic"
        In this example, we want to record data from topics in TopicGroup
        "All" from "domain0." -->

       <record_group name="RecordAll">
           <!-- specify which domains to record from -->
           <domain_ref><element> domain0 </element></domain_ref>

           <!-- specify which topics to record -->
           <topic_ref><element> All </element></topic_ref>
       </record_group>
    </recorder>
</dds>
```

Example configuration files are provided in the **examples/record** directory:

- ❑ **simple_config.xml**

  With this configuration, the *Record* tool will record all fields from all topics in a specified domain (domain ID 0).

- ❑ **advanced_config.xml**

  With this configuration, the *Record* tool will record:

  - The 'x' and 'y'' fields from all Topics named Square in domains 0 and 1.
  - The 'color' field from all Topics in domains 0 and 1.

- ❑ **remote_shell.xml**

  This configuration file provides a configuration that can be used with the tutorial found in the *Recording Service Getting Started Guide* to learn about how to modify the *Record* tool while it is running.

4.2.1 Configuration File Syntax

Recording Service follows the same XML syntax rules as *Connext*. Please see the *RTI Core Libraries and Utilities User's Manual* for details.

4.2.2 Supported Data Types

As you will see in the following sections, each property that can appear in the configuration file uses a specific data type. The *Record* tool converts between the value string in the XML file and the specified type. Table 4.1 lists the supported types and the mappings used by the *Record* tool.

Table 4.1 **Property Value Data Types**

Type	Format	Notes
char and *octet* sequences and arrays	Compact form	
DDS_Boolean	yes,1,true,on: TRUE no,0,false,off: FALSE	These values are not case sensitive.
DDS_Enum	A string	Enum values are not case sensitive. Legal values are those listed for the property in the C API Reference HTML documentation for the *RTI Core Libraries and Utilities* .

Table 4.1 **Property Value Data Types**

Type	Format	Notes
DDS_Long	-2147483648 - 2147483647 0x80000000 - 0x7fffffff	A 32-bit signed integer. You may include the following unit designations: KB — 2^10 kB — 10^3 MB — 10^6 GB — 10^9 KiB — 2^10 MiB — 2^20 GiB — 2^30 For example, 100 kB is a legal value, meaning 100,000.
DDS_ UnsignedLong	0 - 4294967296 0 - 0xffffffff	A 32-bit unsigned integer. You may include the following unit designations: KB — 2^10 kB — 10^3 MB — 10^6 GB — 10^9 KiB — 2^10 MiB — 2^20 GiB — 2^30 For example, 100 kB is a legal value, meaning 100,000.

Table 4.1 **Property Value Data Types**

Type	Format	Notes

DDS_QosPolicy	See the *RTI Core Libraries and Utilities C* API Reference HTML documentation for the structure of each QoS policy, and the *RTI Core Libraries and Utilities User's Manual's* chapter on Configuring QoS with XML.	Each field in each QoS policy structure has a corresponding tag. The tag is the same as the field name in the *RTI Core Libraries and Utilities* C API.
		For enumerations, the legal constants are those defined for the *Connext* C API.
		For example:
		`<subscriber_qos>`
		`  <presentation>`
		`    <access_scope>`
		`      DDS_TOPIC_PRESENTATION_QOS`
		`    </access_scope>`
		`  </presentation>`
		`  <partition>`
		`     <name>`
		`       <element> rti </element>`
		`     </name>`
		`  </partition>`
		`</subscriber_qos>`
		The above configuration will set (a) the Presentation QoS policy's *access_scope* field to DDS_TOPIC_PRESENTATION_QOS and (b) the Partition QoS policy's *name* field to "rti". (*name* is a sequence of strings, which requires using the <element> tag, also described in this table.)

FileSize	64 bit integer	You may include the following unit designations: kB — 10^3 MB — 10^6 GB — 10^9 KB — 2^10 TB — 10^12 KiB — 2^10 MiB — 2^20 GiB — 2^30 TiB — 2^40 For example, 100 kB is a legal value, meaning 100,000.
String	UTF-8 character string	All leading and trailing spaces are ignored between two tags.

4.3 General Properties for the Record Tool

Table 4.2 describes optional properties that control the *Record* tool's main module.

Table 4.2 **General Properties**

Property	Syntax	Description

auto_start	`<auto_start>`
	`  DDS_Boolean`
	`</auto_start>`
		Whether or not the *Record* tool should start recording data when it is started. This option is mostly useful if the *Record* tool is usually controlled remotely.
		Default: True

verbosity	`<verbosity>`
	` DDS_Long`
	`</verbosity>`
		The verbosity is a bit-map that specifies what type of logging information should be printed.
		The verbosity may be:
		0: silent (Core Libraries and the *Record* tool)
		1: errors (Core Libraries and the *Record* tool) (default)
		2: warnings (the *Record* tool only)
		3: warnings (Core Libraries and the *Record* tool)
		4: information (the *Record* tool only)
		5: tracing (the *Record* tool only)
		6: tracing (Core Libraries and the *Record* tool)

4.4 Database (Output File) Properties

Table 4.3 describes the Database properties. All database properties are optional except database_name. All database properties must be specified within **<record_database>** and **</record_database>** tags.

The *Record* tool stores data in a set of SQL database files. (Note, however, that you do *not* need to install any database software to use the *Record* tool.)

[**Note:** Replaying data from a set of files is not supported. This holds true for both *Recording Console* and the *Replay* tool. They can only replay data from one file at a time.]

A *fileset* is a named collection of file segments which belong to the same recording session. Each of these file segments contains discovery and user-data, and the format is determined by SQLite.

Table 4.3 **Database Properties**

Properties under <record_database>	Syntax	Description

create_index	`<create_index>`
	`DDS_Boolean`
	`</create_index>`
		Specifies whether or not the *Record* tool will index the database on the **reception_timestamp** column to allow for efficient replay. Disable for maximum recording performance.
		Default: False

database_name	`<database_name>`
	`String`
	`</database_name>`
		Required.
		The name of the fileset used to store recorded data.
		The *Record* tool appends a set number and a segment number to the filename.
		Default: undefined
		Example:
		`<database_name>`
		`myfile`
		`</database_name>`

flush_period	`<flush_period>`
	`DDS_Long`
	`</flush_period>`
		Specifies how often (in seconds) to flush data to disk.
		Note: increasing this value causes the *Record* tool to use additional memory.
		Default: 1 second
		Minimum: 1 second

max_file_segments	`<max_file_segments>`
	`DDS_Long`
	`</max_file_segments>`
		Specifies how many file segments may be created.
		Each time the **max_file_size** limit is reached for a file segment, a new file is created if this number of segments has not been exceeded.
		Default: 1
		Example:
		`<max_file_segments>`
		`100`
		`</max_file_segments>`

Table 4.3 **Database Properties**

Properties under <record_ database>	Syntax	Description

max_file_size	<max_file_size>
	FileSize
	</max_file_size>
		Specifies the maximum size for a file segment.
		The *Record* tool records data to one or more files. This property specifies the maximum file size. This is not an absolute value, but a threshold value. As soon as the threshold is exceeded, no more data is written to file.
		Default: 2 GB
		Maximum: imposed by the operating system
		Example:
		`<max_file_size>`
		`1 GB`
		`</max_file_size>`

overwrite	<overwrite>
	DDS_Boolean
	</overwrite>
		Specifies whether or not the *Record* tool should delete all existing file segments in the fileset before it starts recording.
		This is useful if you want to reuse a data-file name between recording sessions, but do not want to keep any old data.
		• True: If the file segments already exist, they are deleted; otherwise, the file segments are created as needed.
		• False: If the file segments already exist, the *Record* tool exits; otherwise, the file segments are created as needed.
		Example:
		`<name>test</name>`
		`<max_file_segments>`
		`4`
		`</max_file_segments>`
		`<overwrite>yes</overwrite>`
		In this case, the *Record* tool will delete test_0_0, test_0_1, and test_0_2 before starting to record to test_0_0.
		Default:False

Table 4.3 **Database Properties**

Properties under <record_ database>	**Syntax**	**Description**

path_separator	`<path_separator>`
	`DDS_Char`
	`</path_separator>`
		Specifies the path separator character that the *Record* tool will use when creating table and column names.
		For instance, table names follows the "TopicName$RecordGroupName$DomainName" convention and fields in Topics uses $ to navigate hierarchical types, such as a$b$c.
		'$' is used as the default path separator instead of the more conventional. ',' because $ does not require quotes when used in SQLite SQL statements.
		For example, to use '_' as the path separator:
		`<path_separator> _ </path_separator>`
		Note: this property cannot be empty.

rollover	`<rollover>`
	`DDS_Boolean`
	`</rollover>`
		Specifies whether or not the *Record* tool should overwrite existing file segments in the fileset once the file size limit (**max_file_size**) has been reached for the last file segment.
		True: the *Record* tool overwrites existing file segments as needed (starting with the first one).
		False: the *Record* tool stops recording data.
		Default: False

Table 4.3 **Database Properties**

Properties under <record_ database>	Syntax	Description

self_contained	`<self_contained>`
	`DDS_Boolean`
	`</self_contained`
		Specifies whether to replicate the necessary discovery information for conversion to other formats. If set to true, the Participant and Publication tables will be replicated when a new database file is opened.
		Default: False
		Example:
		`<self_contained>`
		`TRUE`
		`</self_contained>`
		Note: When using self-contained database files, the **locator_filter** column of the Publication Built-in Topic Data will not be replicated. The Subscription Built-in Topic Data will also not be replicated. This is done to minimize the overhead of replication when opening a new database file.

segment_ number	`<segment_number>`
	`DDS_Long`
	`</segment_number>`
		Specifies the first segment to use in the fileset.
		If the segment number is >= 0, that is the first segment number in the fileset.
		Default: -1. The next available segment number will be used, starting at 0.
		Note: the set number is determined first, then the segment number.

set_number	`<set_number>`
	`DDS_Long`
	`</set_number>`
		Specifies the set number to use in the fileset.
		If set_number is >= 0, that specific fileset number is used. In this case, the <overwrite> property takes effect.
		Default: -1. The next available set number will be used, starting at 0.

The *Record* tool uses a fixed file-naming scheme:

name_set-number_segment-number

Where:

❏ *name* is the base filename for the fileset, specified in the configuration file with the **<name>** property.

- *set-number* is an integer identifying the fileset, specified in configuration file with the **<set-number>** property.
- *segment-number* is an integer identifying the file-segment within the fileset. The first segment number to use, and the maximum number of segments are specified in the configuration file with the **<segment_number>** and **<max_file_segments>** properties, respectively.

For example: mydata_5_3 means this file belongs to fileset 5 and is the 3rd segment in that fileset.

The maximum size of a file segment, whether to overwrite existing files, and whether to overwrite the oldest file can all be set in the configuration file.

4.5 Domain Properties

Table 4.4 describes the Domain properties. All Domain properties are optional.

Domain properties must be specified inside **<domain name="String">** and **</domain>** tags. If you want to use a RecordGroup (Section 4.7), you must assign a domain name with these tags, even if you do not specify any domain properties (because the domain name is needed in the RecordGroup's domain_ref property).

You may specify more than one Domain. Each one must have a unique name, with its own **<domain name="String">** and **</domain>** tags.

Note: Transports are configured through the Property QoS under the participant_qos tag.

For example, the following creates a Domain named "mydomain" using domain ID 68. The data will be recorded in serialized format. The DomainParticipant will use default QoS settings, except for the Discovery QoS policy's *accept_unknown_peers* field:

```
<domain name="mydomain">
    <domain_id> 68 </domain_id>
    <deserialize_mode>
        RTIDDS_DESERIALZEMODE_NEVER
    </deserialize_mode>
    <participant_qos>
       <discovery>
         <accept_unknown_peers> false</accept_unknown_peers>
       </discovery>
    <participant_qos>
</domain>
```

Table 4.4 **Domain Properties**

Properties under <domain>	Syntax	Description

deserialize_ mode	`<deserialize_mode>`
	`  DDS_Enum`
	`</deserialize_mode>`
		Determines how topic data is stored in a database (serialized or deserialized).
		The following values are allowed:
		• RTIDDS_DESERIALIZEMODE_AUTOMATIC Deserialize data if possible, otherwise store data in serialized format.
		• RTIDDS_DESERIALIZEMODE_NEVER Do not deserialize the data; store data in serialized format.
		• RTIDDS_DESERIALIZEMODE_ALWAYS Only store data if it can be deserialized first.
		Default: RTIDDS_DESERIALIZEMODE_NEVER
		See Recording Large User Data Types (Section 4.5.2).

domain_id	`<domain_id>`
	`  DDS_Long`
	`</domain_id>`
		Sets the domain ID.
		Default: 0

participant_qos	`<participant_qos>`
	`  DDS_QosPolicy`
	`</participant_qos>`
		Configures the DomainParticipant's QoS policies.
		Default: default DomainParticipant QoS settings
		See the *RTI Core Libraries and Utilities User's Manual* for details. (See the chapter on Configuring QoS with XML.)

4.5.1 Enabling Monitoring Library in the Record Tool

RTI Monitoring Library enables *Connext* applications to provide monitoring data. The monitoring data can be visualized with *RTI Monitor*, a separate GUI application that can run on the same host as *Monitoring Library* or on a different host. *Recording Service* is statically linked to *Monitoring Library* (you do not have to install it separately).

To enable monitoring in the *Record* tool, modify the participants' QoS in the XML configuration to include the **rti.monitor.library** property with a value of **rtimonitoring**. For example:

```
<domain name="domain0">
    <participant_qos>
        <property>
            <value>
```

```
                    <element>
                        <name>rti.monitor.library</name>
                        <value>rtimonitoring</value>
                        <propagate>false</propagate>
                    </element>
                </value>
            </property>
        </participant_qos>
        <domain_id>0</domain_id>
    </domain>
```

See also: Enabling Monitoring Library with Replay (Section 7.4.1).

4.5.2 Recording Large User Data Types

When the *Record* tool records serialized user data, each primitive type in the topic's data structure will have its own column in the table. The maximum number of columns is approximately 1,950.

Therefore, if you have a data-type that would require more than 1,950 columns, you must set the deserialize_mode property to RTIDDS_DESERIALIZEMODE_NEVER. (Disregarding this limit will cause recording to fail.)

Note: *Each primitive type is considered a column*. For example, the following would require 3,000 columns:

```
long Array[3000];
```

As another example, the following would require separate columns for y[0].x.a,y[0].x.b,y[1].x.a,y[1].x.b, etc.

```
struct X {
        long a;
        long b;
};
struct Y {
        X x;
};
struct Z {
        Y y[10];
}
```

4.6 TopicGroup Properties

A TopicGroup is an *optional* logical collection of Topics. If you are not going to have a RecordGroup in the configuration file, you do not need a TopicGroup. (See Section 4.7.)

Table 4.5 describes the TopicGroup properties.

TopicGroup properties must be specified inside **<topic_group name="String">** and **</topic_group>** tags.

The following properties are required:

❏ field_expr

❏ shared_table

❏ topics

Table 4.5 **TopicGroup Properties**

Properties under <topic_ group>	Syntax	Description

auto_ detect_ reliability	`<auto_detect_reliability>`
	DDS_Boolean
	`</auto_detect_reliability>`
		If set to true, use the same reliability as the Publisher of the matched Topic.
		Default: False

compact_ char_array	`<compact_char_array>`
	DDS_Boolean
	`</compact_char_array>`
		Store array of char in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large.
		Default: True

compact_ char_ sequence	`<compact_char_sequence>`
	DDS_Boolean
	`</compact_char_sequence>`
		Store sequence of char in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large.
		Default: True

Table 4.5 **TopicGroup Properties**

Properties under <topic_ group>	**Syntax**	**Description**

compact_ octet_array	`<compact_octet_array>`
	`    DDS_Boolean`
	`</compact_octet_array>`
		Store array of octet in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large.
		Default: True

compact_ octet_ sequence	`<compact_octet_sequence>`
	`    DDS_Boolean`
	`</compact_octet_sequence>`
		Store sequence of octet in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large.
		Default: True

datareader _qos	`<datareader_qos>`
	`    DDS_DataReaderQos`
	`</datareader_qos>`
		Specifies the QoS settings for all DataReaders created for this TopicGroup.
		A DataReader is created for each discovered Topic that matches topic_expr. All the DataReaders for the TopicGroup will use the same set of QoS policies. You can specify all of the QoS policies with the datareader_qos property.
		See the *RTI Core Libraries and Utilities User's Manual* for more information. (See the chapter on Configuring QoS with XML.)
		See the *RTI Core Libraries and Utilities User's Manual* for details. (See the chapter on Configuring QoS with XML.)

exemption	`<exemption>`
	`    POSIX fn expressions`
	`</exemption>`
		Specifies a comma-separated list of expressions that should *not* be recorded.
		Default: Nothing is exempt

Table 4.5 **TopicGroup Properties**

Properties under <topic_ group>	Syntax	Description

field_expr	`<field_expr>`
	`  POSIX fn expressions`
	`</field_expr>`
		Required.
		A list of comma-separated POSIX expressions that specify which fields in the Topics to record. (The Topics are specified with <topics>, see Table 4.6, "Topics Properties".)
		This parameter is ignored when recording serialized data.

include_ meta_ columns	`<include_meta_columns>`
	`    DDS_Boolean`
	`</include_meta_columns>`
		In the database, every sample is stored alongside all its sample information. If this property is set to FALSE, the sample is stored in a serialized way, with no sample information attached to it.
		Setting this to FALSE (not the default) saves storage space. When set to FALSE, less columns are created in the SQLite database. The columns in this database are often filled with repetitive data. So this option can save space and execution time when these requirements are critical.
		Default value: True

Table 4.5 **TopicGroup Properties**

Properties under <topic_ group>	**Syntax**	**Description**

shared_table	`<shared_table>`
	    `DDS_Boolean`
	`</shared_table>`
		Required.
		Specifies whether the tables of recorded data are shared or exclusive.
		The *Record* tool stores Topic data in tables; those tables can be Shared or Exclusive. An Exclusive table means that each Topic recorded in a RecordGroup is stored in its own table. The name of the table follows the convention:
		`TopicName$RecordGroupName$DomainName`
		(You can change the '$' separator with the path_separator database property.)
		Thus, two topics with the same name but from two different TopicGroups are stored in separate tables.
		A shared table means that Topics with the same name are stored in the same table, regardless of where it was recorded from. In this case the table has an additional column, '**table_prefix**', which stores the table prefix in the form: RecordGroupName$DomainName.
		Default: False (exclusive)

topics	`<topics>`
	  `<topic_expr>`
	    `POSIX fn expression`
	  `</topic_expr>`
	`</topics>`
		Required.
		Specifies a topic expression and any exemptions to that expression. See Table 4.6, “Topics Properties”.

Table 4.6 **Topics Properties**

Properties under <topics>	Syntax	Description

topic_expr	`<topic_expr>`
	`POSIX fn expression`
	`</topic_expr>`
		Required.
		A comma-separated list of POSIX expressions that specify the names of Topics to be included in the TopicGroup.
		The syntax and semantics are the same as for Partition matching.
		Default: Null

For example, the following creates a TopicGroup called AllTopics, which will include all discovered Topics. From those Topics, all fields will be recorded. This example does not specify the optional datareader_qos property, so it will use default DataReader QoS settings:

```
<topic_group name="AllTopics">
    <topics>
        <topic_expr> * </topic_expr>
    </topics>
    <field_expr> * </field_expr>
</topic_group>
```

This next example creates a TopicGroup called ColorsOfSquares that will only include Topics named “Square.” For the recorded Topics, only the “color” field will be recorded. The DataReaders for the matching Topics will have default QoS settings, except that the Reliability QoS’s *kind* will be DDS_RELIABLE_RELIABILITY_QOS:

```
<topic_group name="ColorsOfSquares">
    <topics>
        <topic_expr> Square </topic_expr>
    </topics>
    <field_expr> color </field_expr>
    <datareader_qos>
        <reliability>
            <kind> DDS_RELIABLE_RELIABILITY_QOS </kind>
        </reliability>
    </datareader_qos>
</topic_group>
```

The following example creates a TopicGroup called AllMinusCircleAndSquare that will include all Topics except “Circle” and “Square.” For the recorded Topics, all fields will be recorded:

```
<topic_group name="AllMinusCircleAndSquare">
    <topics>
       <topic_expr> * </topic_expr>
       <exemption> Circle, Square </exemption>
    </topics>
    <field_expr> * </field_expr>
</topic_group>
```

Notes:

- ❑ Topics are never removed from a TopicGroup. The resources used to create DataReaders for discovered Topics are not released if/when the Topics are deleted.
- ❑ The *Record* tool will ignore Topics published with a type that conflicts with an already discovered type.

4.7 RecordGroup Properties

A RecordGroup is a binding between a TopicGroup and a Domain. It controls which TopicGroup members are recorded for each Domain. Any Topic that is part of a TopicGroup in the RecordGroup is recorded from the specified Domains.

RecordGroups are optional. If you do not create one, the *Record* tool will not record any data. This is useful if you want to start the *Record* tool in "stand-by" mode—then you can use remote access (see Section 4.8) to switch to a different configuration file (one that does have a RecordGroup) and start recording.

Table 4.7 describes the RecordGroup properties. Notice that domain_ref and topic_ref are required.

Table 4.7 **RecordGroup Properties**

Properties under <record_group>	**Syntax**	**Description**

domain_ref	`<domain_ref>`
	`StringSequence`
	`</domain_ref>`
		Required.
		Specifies a sequence of references to domains.
		Default: Null

Table 4.7 **RecordGroup Properties**

Properties under <record_group>	Syntax	Description

subscriber_qos	`<subscriber_qos>`
	`DDS_SubscriberQos`
	`</subscriber_qos>`
		Configures the Subscriber used by the RecordGroup.
		Default: default Subscriber QoS settings
		See the *RTI Core Libraries and Utilities User's Manual* for details. (See the chapter on Configuring QoS with XML.)

topic_ref	`<topic_ref>`
	`StringSequence`
	`</topic_ref>`
		Required.
		Specifies a sequence of references to TopicGroups.
		Default: Null

RecordGroup properties must be specified inside **<record_group name = "String">** and **</record_group>** tags. The name that you assign ("String") will be used in the table name(s) in the database (output) file(s).

For example, the following creates a RecordGroup called RecordAll, which will include all members of TopicGroup All that are discovered on Domain MyDomain. This example does not specify the optional subscriber_qos property, so it will use default Subscriber QoS settings:

```
<record_group name="RecordAll">
   <topic_ref>
       <element> AllTopics </element>
   </topic_ref>
   <domain_ref>
       <element> MyDomain </element>
   </domain_ref>
</record_group>
```

Note: A RecordGroup can refer to multiple domains and multiple TopicGroups. However, a RecordGroup will only record one of each matching Topic from a Domain. If multiple matches occur, only the first one will be recorded. (If you need to record the same Topic from the same domain using different QoS policies, you should use different TopicGroups and RecordGroups.)

4.8 Remote Access Properties

As you will see in Chapter 5: Accessing the Record Tool from a Remote Location, you can create a *Connext* application that can remotely control the *Record* tool.

By default, Remote Access is turned off in the *Record* tool for security reasons.

The Remote Access section of the configuration file is used to enable Remote Access and configure its behavior. A Remote Access section is not required in the configuration file.

The remote application can send commands to the *Record* tool that will:

- ❑ Start/stop recording.
- ❑ Shutdown the *Record* tool.
- ❑ Reconfigure the *Record* tool.

Table 4.8 describes the Remote Access properties. All Remote Access properties must be specified inside **<remote_access>** and **</remote_access>** tags. All remote access properties are optional unless otherwise noted.

Table 4.8 **Remote Access Properties**

Properties under <remote_ access>	Syntax	Description

accept_ broadcast_ commands	`<accept_broadcast_commands>`
	`DDS_Boolean`
	`</accept_broadcast_commands>`
		Specifies if the *Record* tool will accept commands that have been broadcast to any *Record* tool or only accept commands addressed specifically to it. (See Chapter 5: Accessing the Record Tool from a Remote Location.)
		Default: True

enabled	`<enabled>`
	`DDS_Boolean`
	`</enabled>`
		Enables or disables remote access to the *Record* tool from another application. (See Chapter 5: Accessing the Record Tool from a Remote Location.)
		Default: False (remote access disabled)

datareader_qos	`<datareader_qos>`
	`DDS_DataReaderQos`
	`</datareader_qos>`
		Configures the QoS for the DataReader created by the *Record* tool's Remote Access module.
		Default: default DataReader QoS settings.

Table 4.8 **Remote Access Properties**

Properties under <remote_ access>	**Syntax**	**Description**

datawriter_qos	`<datawriter_qos>`
	    `DDS_DataWriterQos`
	`</datawriter_qos>`
		Configures the QoS for the DataWriter created by the *Record* tool's Remote Access module.
		Default: Default DataWriter QoS settings.

distributed_ logger	`<distributed_logger>`
	    `Distributed Logger Properties`
	`</distributed_logger>`
		Configures *RTI Distributed Logger.*
		See Enabling RTI Distributed Logger in the Record Tool (Section 4.8.1).

publish_ status_ period	`<publish_status_period>`
	    `DDS_Long`
	`</publish_status_period>`
		Specifies, in seconds, the period between each status message sent by the *Record* tool.
		Default: 1. Minimum value: 1.

publisher_qos	`<publisher_qos>`
	    `DDS_PublisherQos`
	`</publisher_qos>`
		Configures the QoS for the Publisher created by the *Record* tool's Remote Access module.
		Default: default Publisher QoS settings.

remote_ access_ domain	`<remote_access_domain>`
	    `String`
	`</remote_access_domain>`
		Required if enabled is true.
		Specifies which domain the *Record* tool will use to enable remote access. Only one domain can be specified.
		Note that this is a String, not a Domain ID. It is the same String used in the <domain name="String"> </domain> line.
		Default: False

subscriber_qos	`<subscriber_qos>`
	    `DDS_QosPolicy`
	`</subscriber_qos>`
		Configures the QoS for the Subscriber created by the *Record* tool's Remote Access module.
		Default: Default Subscriber QoS settings.

4.8.1 Enabling RTI Distributed Logger in the Record Tool

The *Record* tool provides integrated support for *RTI Distributed Logger*.

When you enable *Distributed Logger*, the *Record* tool will publish its log messages to *Connext*. Then you can use *RTI Monitor*[1] to visualize the log message data. Since the data is provided in a *Connext* topic, you can also use *rtiddsspy* or even write your own visualization tool.

To enable *Distributed Logger*, modify the *Record* tool's XML configuration file. In the <remote_access> section, add the <distributed_logger> tag as shown in the example below.

```
<remote_access>
    <enabled>true</enabled>
    <remote_access_domain>adminDomain</remote_access_domain>
    <distributed_logger>
        <enabled>true</enabled>
    </distributed_logger>
</remote_access>
```

There are more configuration tags that you can use to control *Distributed Logger's* behavior. For example, you can specify a filter so that only certain types of log messages are published. For details, see the *RTI Distributed Logger Getting Started Guide.*

4.9 Domain Type Configuration

The top-level tag <domain_type_config> allows you to pass type configuration information to the *Record* and *Convert* tools in the form of XML type-configuration files.

[1] *RTI Monitor* is a separate GUI application that can run on the same host as your application or on a different host.

Table 4.9 describes the Domain Type Config properties. All Domain Type Config properties are optional.

Table 4.9 **Domain Type Configuration Properties**

Properties under <domain_ type_config>	Syntax	Description

domain_group	`<domain_group>`
	`  <element>`
	`    ` *`Domain Group`*
	`    ` *`Properties`*
	`  </element>`
	`</domain_group>`
		A list of type configuration elements associated with specific Recorder domain definitions (see Section 4.5). These type configuration elements can be repeated.
		For more details in the contents of this tag See Table 4.10, “Domain Group Configuration Properties”.

Table 4.10 **Domain Group Configuration Properties**

Properties under <domain_ group>	Syntax	Description

domain_filter	`<domain_filter>`
	`  <element>`
	`    ` *`POSIX fn expression`*
	`  </element>`
	`</domain_filter>`
		A list of POSIX expressions that specify the names of the Recorder domain definitions for which the type definitions specified will apply. The element tag can be repeated. The POSIX expressions have to match the name attribute of any of the domain definitions as defined in Section 4.5.

type_config	`<type_config>`
	`  ` *`XML Properties`*
	`</type_config>`
		Specific XML type configuration properties for the domains specified by the list of elements in the domain_filter tag seen above.
		See Table 4.11, “Type Config Properties”.

Table 4.11 **Type Config Properties**

Properties under <type_config>	Syntax	Description

xml	`<xml>`
	`XML Type Configuration Properties`
	`</xml>`
		Allows you to specify XML type-configuration files, the path in which to find them, and other properties related to the registration of the types with the DomainParticipants.
		See Table 4.12, "XML Type Configuration Properties".

Table 4.12 **XML Type Configuration Properties**

Properties under <xml>	Syntax	Description

file_group	`<file_group>`
	`<element>`
	`File Group Properties`
	`</element>`
	`<file_group>`
		Allows you to specify XML file groups; each of these files contain type definitions to be used by the *Record* tool. The element tag can be repeated.
		See Table 4.13, "File Group Properties".

max_sequence	`<max_sequence>`
	`Integer`
	`</max_sequence>`
		The default sequence size, in case there are unbounded sequences in the type definitions specified in the any of the files specified in any of the file groups.

max_string	`<max_string>`
	`Integer`
	`</max_string>`
		The default string size, in case there are unbounded strings in the type definitions specified in any of the files specified in any of the file groups.

path	`<path>`
	`<element>`
	`Path`
	`</element>`
	`</path>`
		A list of the paths (relative or absolute) to be used when searching for the XML type definition files. The <element> tag can be repeated.

register_ top_level	`<register_top_level>`
	`Boolean`
	`</register_top_level>`
		Whether or not to register the top-level types found in the type definitions with their canonical names. Do this with any of the files defined in any of the file groups. Default: TRUE.

Table 4.13 **File Group Properties**

Properties under <file_group> <element>	Syntax	Description

file_name	`<file_name>`
	`<element>`
	`File Name`
	`</element>`
	`</file_name>`
		A list of file name strings, specifying files containing the type definitions. The element tag can be repeated.

type	`<type>`
	`<element>`
	`Type Registration Properties`
	`</element>`
	`</type>`
		Specific type properties. A list of type registration properties used to define how a type found in the files has to be registered by Recorder in the DomainParticipants. See Table 4.14, "Type Registration Properties"

max_sequence	`<max_sequence>`
	`Integer`
	`</max_sequence>`
		The default sequence size in case there are unbounded sequences in the type definitions specified in the any of the files specified in this specific file group.

max_string	`<max_string>`
	`Integer`
	`</max_string>`
		The default string size in case there are unbounded strings in the type definitions specified in any of the files specified in this specific file group .

register_ top_level	`<register_top_level>`
	`Boolean`
	`</register_top_level>`
		Whether or not to register the top-level types found in the type definitions with their canonical names. Do this with any of the files found in this specific file group. The value of this setting overrides the value of the upper-level setting (see Table 4.12, "XML Type Configuration Properties"). Default: TRUE.

Table 4.14 **Type Registration Properties**

Properties <under <type> <element>	Syntax	Description

type_name	`<type_name>`
	`Type Name string`
	`</type_name>`
		A string representing the canonical type name as defined in the XML type definition for the type.

Table 4.14 **Type Registration Properties**

Properties <under <type> <element>	Syntax	Description

registered_ type_name	`<registered_type_name>`
	`    <element>`
	`        Registration`
	`        name string`
	`    </element>`
	`</registered_type_name>`
		When registering the type with the DomainParticipant, this setting defines a list of names to register the type with. The element tag can be repeated.

register_top_ level	`<register_top_level>`
	`    Boolean`
	`</register_top_level>`
		Whether or not to register this type's canonical name (as defined in tag type_name) with the DomainParticipant. This name is registered alongside any of the registration type names defined in tag registered_type_name above. Default: TRUE.

4.10 Recording Service Integration with Extensible Types

Recording Service includes partial support for the "Extensible and Dynamic Topic Types for DDS" specification from the Object Management Group (OMG)[1]. This section assumes that you are familiar with Extensible Types and you have read the *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types.*

This support allows systems to define data types in a more flexible way, and to evolve data types over time without giving up portability, interoperability, or the expressiveness of the DDS type system.

With Extensible Types, different type definitions for the same type name may be discovered by the *Record* tool during execution. The tool’s default behavior is to register the first type definition (type code) found for a type.

If you have the *Connext Core Libraries and Utilities*, you can learn more in the *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types*[2].

[1] http://www.omg.org/spec/DDS-XTypes/

[2] <*Connext installation directory*>/ndds.<*version*>/doc/pdf/RTI_CoreLibrariesAndUtilities_GettingStarted_ExtensibleTypesAddendum.pdf)

- *Recording Service* can subscribe to topics associated with final and extensible types.
- A topic "T" within a recording service domain (**<domain>**) can be associated with at most one version of a type. To record more than one version, you will have to use multiple recording service domains.
- Users can pre-register a set of types in a recording service domain by providing an XML description of the types. The XML description supports structure inheritance (described in the *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types*).
- The TypeConsistencyEnforcementQosPolicy can be specified on a per-topic-group (**<topic_group>**) basis, in the same way as other QoS policies. Conversions between extensible types are allowed if the TypeConsistencyEnforcementQosPolicy's **kind** is set to ALLOW_TYPE_COERCION.

4.10.1 Selecting a Type Version For a Topic "T" In a Recording Domain

A topic "T" within a recording service domain (**<domain>**) can be associated with at most one version of a type. By default, *Recording Service* will use the type associated with the first discovered *DataWriter* on topic "T". This makes the selection of the type non-deterministic since it depends on the order in which *DataWriters* are discovered. To resolve this problem, you can provide the type in the XML configuration using the **<domain_type_config>**. For example:

```
<domain_type_config>
  <domain_group>
    <element>
      <domain_filter>
        <element>domain0</element>
      </domain_filter>
      <type_config>
        <xml>
          <file_group>
            <element>
              <file_name>
                <element>ShapeType.xml</element>
              </file_name>
              <type>
                <element>
                  <type_name>ShapeTypeExtended</type_name>
                  <registered_type_name>
                    <element>ShapeType</element>
                  </registered_type_name>
                  <register_top_level>false</register_top_level>
                </element>
              </type>
            </element>
          </file_group>
          <path>
            <element>.</element>
          </path>
        </xml>
      </type_config>
    </element>
  </domain_group>
</domain_type_config>
```

The type itself is described in the file **ShapeType.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<types>
  <enum name="ShapeFillKind" bitBound="32">
    <enumerator name="SOLID_FILL"/>
    <enumerator name="TRANSPARENT_FILL"/>
    <enumerator name="HORIZONTAL_HATCH_FILL"/>
    <enumerator name="VERTICAL_HATCH_FILL"/>
  </enum>
  <struct name="ShapeType">
    <member name="color" stringMaxLength="128"
      type="string" key="true"/>
    <member name="x" type="long"/>
    <member name="y" type="long"/>
    <member name="shapesize" type="long"/>
  </struct>
  <struct name="ShapeTypeExtended" baseType="ShapeType">
    <member name="fillKind"
     type="nonBasic" nonBasicTypeName="ShapeFillKind"/>
    <member name="angle" type="float"/>
  </struct>
</types>
```

In the above example, *Recording Service* will register the type ShapeTypeExtended with the DomainParticipant associated with “domain0” using the registered type name “ShapeType”.

When *Recording Service* discovers the first DataWriter on topic “T”, it will try to create the topic using the DataWriter’s registered type name. If the type name has not been registered with the recording service’s DomainParticipant yet, the new topic will be associated with the discovered type. If the type name is already registered the new topic will use the registered type.

Notice that in *Connext* 5.0, the association between topic and type is not explicit; it is done using the registered type name of the first DataWriter discovered publishing topic "T". To guarantee that a recording domain uses a specific type version for a topic "T", you must register that version using all the registered type names that the DataWriters publishing topic "T" may use. For example, in the previous example, if we have a DataW*riter* using the registered type name "ShapeType" and a different *DataWriter* using "ShapeType2", the configuration would be something like this:

```
<domain_type_config>
  <domain_group>
    <element>
      <domain_filter>
        <element>domain0</element>
      </domain_filter>
      <type_config>
        <xml>
          <file_group>
            <element>
              <file_name>
                <element>ShapeType.xml</element>
              </file_name>
              <type>
                <element>
                  <type_name>ShapeTypeExtended</type_name>
                  <registered_type_name>
                    <element>ShapeType</element>
                    <element>ShapeType2</element>
                  </registered_type_name>
                  <register_top_level>false</register_top_level>
                </element>
              </type>
            </element>
          </file_group>
          <path>
            <element>.</element>
          </path>
        </xml>
      </type_config>
    </element>
  </domain_group>
</domain_type_config>
```

4.10.2 Example

Let’s consider an example based on *RTI Shapes Demo* (described in the *Recording Service Getting Started Guide*). You can use *Shapes Demo* to publish and/or subscribe to either the "ShapeType" data type or the "ShapeExtended" data type. ShapeExtended includes the same data as ShapeType, plus two more fields: FillKind and Angle.

Suppose *Shapes Demo* is publishing the base ShapeType in a domain and the *Record* tool is subscribing to ShapeType in the same domain. The *Record* tool will register and record data of type ShapeType. If a different instance of *Shapes Demo*—one that uses the ShapeExtended type—starts publishing in the same domain, the *Record* tool will record new data from this second instance of *Shapes Demo*. (Both types are compatible because one extends the other.) However, the *Record* tool will only record the fields corresponding to the base ShapeType, because that is the registered type. The *Record* tool will ignore the extra fields in the ShapeExtendedType (FillKind and Angle).

As described in Selecting a Type Version For a Topic “T” In a Recording Domain (Section 4.10.1), this default behavior is non-deterministic regarding the types being recorded. That is, you have no control over what type version will be recorded. To change this behavior, you can change the *Record* tool’s configuration and provide the XML description of the desired type version via XML. We illustrate this with the following example, whose objective is to make sure that the *Record* tool records the type ShapeTypeExtended, no matter which types are being published. To do so, the *Record* tool needs a file that defines the type to be recorded, such as the following file named **ShapeType.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<types>
  <enum name="ShapeFillKind" bitBound="32">
    <enumerator name="SOLID_FILL"/>
    <enumerator name="TRANSPARENT_FILL"/>
    <enumerator name="HORIZONTAL_HATCH_FILL"/>
    <enumerator name="VERTICAL_HATCH_FILL"/>
  </enum>

  <struct name="ShapeType">
    <member name="color" stringMaxLength="128"
     type="string" key="true"/>
    <member name="x" type="long"/>
    <member name="y" type="long"/>
    <member name="shapesize" type="long"/>
  </struct>

  <struct name="ShapeTypeExtended" baseType="ShapeType">
```

```
    <member name="fillKind" type="nonBasic"
     nonBasicTypeName="ShapeFillKind"/>
    <member name="angle" type="float"/>
  </struct>
</types>
```

Using the following configuration, the *Record* tool will always record the ShapeTypeExtended by using the XML type definition in **ShapeType.xml**. With the configuration below, no information from a *DataWriter* using ShapeTypeExtended will be lost.

```
<dds>
   <!-- Configuration records the extended ShapeType in domain 0 -->
   <recorder name="extendedShapeTypeRecorder">

      <!-- Specify where to store the recorded data. -->
      <recorder_database>
         <database_name> shapes.dat </database_name>
      </recorder_database>

      <!-- Specify XML file that defines type structure
            for ShapeType -->
      <domain_type_config>
         <domain_group>
            <element>
               <domain_filter>
                  <!-- Specify which domains use this
                       type definition -->
                  <element>domain0</element>
               </domain_filter>
               <type_config>
                  <xml>
                     <file_group>
                        <element>
                           <file_name>
                              <!-- Name of file that defines
                                   ShapeType. More than one file
                                   can be included if needed.-->
                              <element>ShapeType.xml</element>
                           </file_name>
                           <type>
                              <element>
                                 <type_name>
                                    ShapeTypeExtended
                                 </type_name>
                                 <registered_type_name>
                                    <element>ShapeType</element>
                           </registered_type_name>
                           <register_top_level>
                              false
                           </register_top_level>
                        </element>
                     </type>
                  </element>
               </file_group>
               <path>
                  <!-- Define a sequence of paths in which
                       to look for the above files -->
                  <element>.</element>
               </path>
            </xml>
         </type_config>
      </element>
   </domain_group>
</domain_type_config>

<!-- Create DomainParticipant in domain 0 with default QoS -->
<domain name="domain0">
   <domain_id> 0 </domain_id>
   <!-- Always deserialize so the type fields are shown in
        columns (for verification purposes) -->
   <deserialize_mode>
     RTIDDS_DESERIALIZEMODE_ALWAYS
   </deserialize_mode>
</domain>

<!-- Create a generic TopicGroup -->
<topic_group name="All">
   <topics>
      <topic_expr> * </topic_expr>
   </topics>
   <field_expr> * </field_expr>
</topic_group>

<!-- Create RecordGroup to record generic TopicGroup
     in domain 0 -->
<record_group name="RecordAll">
   <!-- specify from which domains to record -->
   <domain_ref>
      <element> domain0 </element>
   </domain_ref>
   <!-- specify which topics to record -->
```

```
         <topic_ref>
            <element> All </element>
         </topic_ref>
      </record_group>
  </recorder>
</dds>
```

The above example assumes that all the *Shapes Demo DataWriters* register their types using the name “ShapeType”.

Chapter 5 Accessing the Record Tool from a Remote Location

Perhaps you want to start/stop the *Record* tool from another machine, or even reconfigure it to change what is being recorded. You can create a *Connext* application that can remotely control the *Record* tool. This chapter explains how.

To control the *Record* tool from a remote location:

1. Configure the *Record* tool to allow remote access (see Remote Access Properties (Section 4.8).
2. Create a *Connext* application using the provided **rtirecord.idl** file. You will use *rtiddsgen* to generate the basics and then add code to send your desired remote commands.

5.1 Overview

If the *Record* tool is configured to allow remote access, it creates a DataReader for a “command” Topic (named RTI_RECORDER_CMD_TOPIC) and a DataWriter for “status” Topic (named RTI_RECORDER_STATUS_TOPIC). So the *Record* tool will write status updates and read commands.

These topics’ types and names are specified in the IDL file, **resource/idl/rtirecord.idl**.

When the *Record* tool detects a remote DataReader and DataWriter of these special topics from the same participant, the *Record* tool will be in a ‘connected’ state, which means it will accept remote commands.

Your remote-access application will use the following constants:

- ❑ **RTI_REMOTECTX_MSG_TYPE** Register a type of this name, as seen in Figure 5.1.
- ❑ **RTI_RECORDER_CMD_TOPIC** Create a DataWriter with this Topic name, as seen in Figure 5.2.
- ❑ **RTI_RECORDER_STATUS_TOPIC** Create a DataReader with this Topic name, as seen in Figure 5.2.

See Remote Control Messages (Section 5.3) for more information.

Figure 5.1 **Registering the Message Type**

```
RTIRemoteCtxMsgTypeSupport_register_type
                    (self->dds_participant, RTI_REMOTECTX_MSG_TYPE);
```

5.2 Establishing a Connection with the Record Tool

To establish a connection with the *Record* tool, your remote-access application needs:

- ❑ 2 Topics (one for commands, one for status)
- ❑ 1 DataReader
- ❑ 1 DataWriter

When creating the DataReader and DataWriter, use the following QoS settings:

- ❑ **history.kind** = DDS_KEEP_ALL_HISTORY_QOS
- ❑ **reliability.kind** = DDS_RELIABLE_RELIABILITY_QOS

Figure 5.2 shows how to create the Entities in your remote-control application using the C API. (A general knowledge of *Connext* is assumed.)

Figure 5.2 **Creating the Required Entities**

```
dds_topic_cmd =
           DDS_DomainParticipant_create_topic(
                  dds_participant, RTI_RECORDER_CMD_TOPIC,
                  RTI_REMOTECTX_MSG_TYPE, &tqos,
                  NULL, DDS_STATUS_MASK_NONE);
dds_topic_status =
           DDS_DomainParticipant_create_topic(
                  dds_participant, RTI_RECORDER_STATUS_TOPIC,
                  RTI_REMOTECTX_MSG_TYPE, &tqos,
                  NULL, DDS_STATUS_MASK_NONE);
...

rqos.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;
rqos.history.kind = DDS_KEEP_ALL_HISTORY_QOS;

dds_reader = (RTIRemoteCtxMsgDataReader*)
        DDS_Subscriber_create_datareader(
                  dds_subscriber,
                  DDS_Topic_as_topicdescription(
                         dds_topic_status), &rqos,
                         NONE, DDS_STATUS_MASK_NONE);

wqos.reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;
wqos.history.kind = DDS_KEEP_ALL_HISTORY_QOS;

dds_writer = (RTIRemoteCtxMsgDataWriter*)
              DDS_Publisher_create_datawriter(
                  dds_publisher, dds_topic_cmd, &wqos,
                  NULL, DDS_STATUS_MASK_NONE);
```

5.3 Remote Control Messages

The *Record* tool exchanges messages with your remote-access application by publishing and subscribing to two special remote-access topics. Both topics use the same message format, shown in Figure 5.3.

Figure 5.3 **Top Level Structure for Remote Control Messages**

```
struct RTIRemoteCtxMsg {
    long destination_mask;
    RTIRemoteCtxAddress destination;
    long msg_id;
    RTIRemoteCtxMsgUnion msg;
};
```

For complete details, see the IDL file, **rtirecord.idl** in the **examples** directory.

destination_mask — A field used by other RTI tools; can be ignored.

destination — The *Record* tool application for which the message is intended. If **accept_broadcast_commands** is turned off, this structure must match that of the *Record* tool. If **accept_broadcast_commands** is turned on, this structure can be a specific destination or all 0's.

msg_id — A user-specified integer that identifies a particular message exchange. When the *Record* tool sends a response to a command, it will include the same **msg_id** that was received in the command.

msg — A union of different message types. The discriminator must be set to one of the message types listed in Table 5.1.

The code fragment in Figure 5.4 shows how to set the message type in the remote-access application.

Depending on the message type, the correct union member must also be filled in. For example, Figure 5.5 shows how to construct a message to the *Record* tool to read a new configuration from a file. In this example, the new configuration is to be read from a file on the same file-system as the *Record* tool.

Figure 5.4 **Assigning a Message Type (C Language)**

```
RTIRemoteCtxMsg *msg;
msg = RTIRemoteCtxMsgPlugin_create_sample();
msg->msg._d = RTI_REMOTECTX_MSG_RECORDER_START;
```

Figure 5.5 **Sending a Command to the Record Tool to Read a New Configuration File**

```
RTIRemoteCtxMsg *msg;
DDS_ReturnCode_t retcode;
Struct DDS_SampleInfo info;
msg =  RTIRemoteCtxMsgPlugin_create_sample();
msg->msg._d = RTI_REMOTECTX_MSG_RECORDER_CONFIGURE;
/* This is the last part of the configuration. If the
 * configuration spans multiple samples, then only the last
 * one should have this set to TRUE.
 */
msg->msg._u.config.final_config = DDS_BOOLEAN_TRUE;

/* Tell the Record Tool that the filename to read from follows
 * in the config_from_string text string
 */
msg->msg._u.config.config_from_file = DDS_BOOLEAN_TRUE;

/* copy the name of file that the Record tool shall read from */
strncpy(msg->msg._u.config.config_from_string,filename,512);

/* copy the configuration name of the <recorder> tag to load */
strncpy(msg->msg._u.config.config_name, cfgname, 512);

/* Send the configuration message to the Record tool.
 * (dds_writer has been created elsewhere)
 */
retcode = RTIRemoteCtxMsgDataWriter_write(
                               dds_writer, msg, &DDS_HANDLE_NIL);
/* check for errors here ... */

while (no response) {
    retcode = RTIRemoteCtxMsgDataWriter_read_next_sample(
                                dds_reader, msg,
                                &info, &DDS_HANDLE_NIL);
    if (retcode != DDS_RETCODE_NO_DATA) {
           /* response received */
    }
    sleep(1);
}
```

Table 5.1 **Messages Types Exchanged Between Record Tool and Remote Access Application**

Direction	Message Type (add prefix: RTI_REMOTECTX_ MSG_)	Description
From: Your *Connext* Remote-Control Application To: The *Record* tool	RECORDER_START	Instructs the *Record* tool to start recording.
	RECORDER_STOP	Instructs the *Record* tool to stop recording.
	RECORDER_ CONFIGURE	Instructs the *Record* tool to reconfigure according to the contents of the message. Stop the *Record* tool before sending this message: If the *Record* tool has already been stopped, it will read the new configuration and restart. It will not automatically start recording unless auto_start (see Table 4.2, "General Properties") is true (the default case). If the *Record* tool has not already been stopped, an error is returned.
	RECORDER_ SHUTDOWN	Instructs the *Record* tool to shutdown and exit.
	RECORDER_ADD	Instructs the *Record* tool to add entities based on the contents of the message.
	RECORDER_DELETE	Instructs the *Record* tool to delete entities based on the contents of the message.
	RECORDER_PAUSE	Instructs the *Record* tool to pause entities based on the contents of the message.
	RECORDER_RESUME	Instructs the *Record* tool to resume recording of previously paused entities based on the contents of the message.
	RECORDER_PING	Instructs the *Record* tool to send the recording model [a] to the Remote Control application.
To: Your *Connext* Remote-Control Application From: The *Record* tool	RECORDER_INFO	When the *Record* tool publishes statistics, it periodically sends out this message type.
	RECORDER_ RESPONSE	Indicates that this message is a response to a command.

[a] The *recording model* is an XML representation of two aspects of the *Record* tool: (1) The configuration model: the XML configuration (similar to the XML configuration file used to configure the *Record* tool.) and (2) The run-time model: an XML description of the entities that have been created based on the configuration. Note that only a a minimal model is returned; the QoS are not returned.

5.4 Using the Example Remote-Access Application—Record Shell

The *Record Shell* is a *Connext* application that can remotely control (start, stop, and reconfigure) the *Record* tool. The *Record Shell* is not meant as a complete solution to remotely controlling the *Record* tool. Its purpose is just to give you an idea of what can be done.

The *Record Shell*, **rtirecsh**, is in **<*install directory*>/bin/<*architecture*>**.

For example, to start the *Record Shell*, enter:

```
cd <install directory>/scripts
rtirecsh -domain <domain ID>
```

Table 5.2 lists the command-line options you can use when starting the *Record Shell*. Once it is running, you can use the commands described in Record Shell's Commands (Section 5.4.1).

Table 5.2 **Record Shell's Command-Line Options**

Command-line Option	Description
-domain <domain ID>	**Required.** Specifies the domain ID (an integer between 0 and 99).
-partition <names>	Specifies an optional, comma-separated list of partition names. This option is necessary if the *Record* tool is configured to enable remote access in a particular partition.
-noUdpv4	Disables the UPDv4 transport.
-updv6	Enables the UDPv6 transport.
-noShmem	Disables the shared memory transport.
-noMulticast	Disables multicast.
-verbosity <mask>	The verbosity is a bit-map that specifies what type of logging information should be printed. The verbosity may be: 0 — No messages 1 — Exceptions (default) 2 — Warnings 4 — Information 7 — All types
-help	Prints version information and a list of options.

5.4.1 Record Shell's Commands

- ❏ add (Section 5.4.1.1)
- ❏ configure (Section 5.4.1.2)
- ❏ delete (Section 5.4.1.3)
- ❏ exit (Section 5.4.1.4)
- ❏ info (Section 5.4.1.5)
- ❏ model (Section 5.4.1.6)
- ❏ pause (Section 5.4.1.7)
- ❏ resume (Section 5.4.1.8)
- ❏ shutdown (Section 5.4.1.9)
- ❏ start (Section 5.4.1.10)
- ❏ status (Section 5.4.1.11)
- ❏ stop (Section 5.4.1.12)

Several of the commands accept a **<model>** argument. A *model* is an XML representation of two aspects of the *Record* tool:

- ❏ The configuration model: the XML configuration (similar to the XML configuration file used to configure the *Record* tool.)
- ❏ The run-time model: an XML description of the entities that have been created based on the configuration.

The format of this XML is the same as the configuration format for the *Record* tool (see Chapter 4: Configuring the Record Tool). **The top-level tag must be <dds> followed by <recorder>**.

Some examples for <model> are:

```
<dds>
    <recorder>
        <record_group name="RecordAll"></record_group>
    </recorder>
</dds>
```

```
<dds>
    <recorder>
        <topic_group>name="RTI Shapes Demo">
            <topics>
                <topic_expr>Square</topic_expr>
                </topics>
                <field_expr>color</field_expr>
            </topic_group>
        </recorder>
    </dds>
```

5.4.1.1 add

This command adds entities to the *Record* tool.

The **add** command has the following format:

```
add <model>
```

5.4.1.2 configure

The *Record* tool can be reconfigured remotely with the configure command. There are two ways to reconfigure the *Record* tool; using a local file or a remote file.

Note that the *Record* tool must be stopped before it can be reconfigured. When the *Record* tool is reconfigured, it will shut down completely. The *Record Shell* will lose its connection with the *Record* tool until the *Record* tool re-establishes remote access. If remote access is not enabled in the new configuration, *Record Shell* will not reconnect to the *Record* tool.

The **configure** command has the following format:

```
configure <cfg_name> [-localfile | -remotefile ] <file>
```

The configuration name **<cfg_name>** is used to find the matching <recorder> tag to load.

❑ **-localfile <filename>**

Example: Assume that you want to the *Record* tool to use a configuration file called **myconfig.xml**, which is local to the *Record Shell*:

```
RTI Record Shell> stop
RTI Record Shell> configure myrecord -localfile myconfig.xml
```

The *Record Shell* will read the contents of **myconfig.xml** and send it to the *Record* tool, which will search for a tag <record name="myrecord">. If **auto_start** (see Table 4.2, "General Properties") is true (the default case), it is not necessary to run the start command to start the *Record* tool. If **auto_start** is false in the new configuration, then issue the start command in the *Record Shell* to start recording:

```
RTI Record Shell> start
```

- ❑ **-remotefile <filename>**

  To configure the *Record* tool with the contents of a file that is local to the *Record* tool, use the **-remotefile <filename>** option.

  For example, assume that you want reconfigure the *Record* tool with a file called **remotemyconfig.xml**, which resides on the same file-system as the *Record* tool.

  ```
  RTI Record Shell> stop
  RTI Record Shell> configure myrecord -remotefile remotemyconfig.xml
  ```

  The *Record* tool will read the contents of **remotemyconfig.xml** and reconfigure with the contents of the tag <record name=”myrecord”>. Depending on the configuration file, it may be necessary to start it:

  ```
  RTI Record Shell> start
  ```

5.4.1.3 delete

This command deletes entities from the *Record* tool.

The **delete** command has the following format:

```
delete <model>
```

5.4.1.4 exit

This command exits the *Record Shell*.

```
RTI Record Shell> exit
```

5.4.1.5 info

This command shows you which *Record* tool session the *Record Shell* is connected to. The output looks similar to this:

```
STATE ..: Connected to [0a0a64fe.006bbe00]
GUID ...: 0a0a64fe.006bbb00
```

- ❑ **STATE** Which DomainParticipant the *Record* tool is connected to (HOSTID.APPID].
- ❑ **GUID** The GUID of the *Record Shell* itself.

5.4.1.6 model

This command prints the current model of the *Record* tool.

```
RTI Record Shell> model
```

5.4.1.7 pause

This command pauses the recording of entities in the *Record* tool.

The **pause** command has the following format:

```
pause <model>
```

5.4.1.8 resume

This command resumes the recording of already paused entities in the *Record* tool.

The **resume** command has the following format:

```
resume <model>
```

5.4.1.9 shutdown

This command causes the *Record* tool to shut down and terminate.

This command can only be issued when the *Record* tool has been stopped.

5.4.1.10 start

The **start** command is used to start the *Record* tool. Note that this command only works after stopping the *Record* tool first, since the tool is started when it is launched.

When the start command is given, the *Record* tool will shut down completely, delete all state and objects and start from scratch. By default, the *Record* tool will create a new file-set each time it is started.

5.4.1.11 status

When the *Record* tool is configured with remote access enabled, it will periodically send its current status. The *Record Shell* stores the most recent status. The current status is displayed with the **status** command:

```
RTI Record Shell> status
```

The output is similar to the following:

```
Version ..........: 1.4.2.v20080528205153
Timestamp ........: Mon Apr 28 20:02:48 2008 (1182222168.58877000)
State ............: STOPPED
Config file ......: simple_config.xml
Database file ....: simple_config.dat_34_3
Received bytes ...: 86653952
Saved bytes ......: 2127872 (2 %)
```

- ❑ **Version** The *Record* tool's version
- ❑ **Timestamp** The timestamp of the *Record* tool when status message was sent.
- ❑ **State** The *Record* tool*'s* state. The following states are possible:
  - IDLE
  - RECORDING
  - STOPPED (the *Record* tool has been stopped and is not recording any user data)
  - RECONFIGURE
  - SHUTDOWN
  - RESTART
  - DOWNLOAD (the *Record* tool is downloading a new configuration)
- ❑ **Config file** The name of the file from which the *Record* tool read its configuration. If the configuration was received with **configure -localfile**, this field is not available.
- ❑ **Database file** The file-segment currently being written to.
- ❑ **Received bytes** Total amount of data that has been written to file.
- ❑ **Saved bytes** Total number of data that has is currently saved to file. Note that if the rollover property is true, then Saved bytes may be less than Received bytes.

5.4.1.12 stop

This command stops the *Record* tool from recording user data.

```
RTI Record Shell> stop
```

5.4.2 Running Multiple Record Tools in the Same Domain

The *Record Shell* can only keep track of one instance of the *Record* tool. To control multiple copies of the *Record* tool in the same domain with the *Record Shell*, run each *Record* tool instance in a separate partition.

For the first instance of the *Record* tool, change the configuration file as follows:

```
<remote_access>
    <enabled> true </enabled>
    <domain> domain0 </domain>
    <subscriber_qos>
         <partition>
              <name>
                   <element> RecordA </element>
```

```
                </name>
           </partition>
      </subscriber_qos>
      <publisher_qos>
           <partition>
                <name>
                     <element> RecordA </element>
                </name>
           </partition>
      </publisher_qos>
   </remote_access>
```

For the second instance of the *Record* tool, change the configuration file as follows:

```
<remote_access>
   <enabled> true </enabled>
   <domain> domain0 </domain>
   <subscriber_qos>
        <partition>
             <name>
                  <element> RecordB </element>
             </name>
        </partition>
   </subscriber_qos>
   <publisher_qos>
        <partition>
             <name>
                  <element> RecordB </element>
             </name>
        </partition>
   </publisher_qos>
</remote_access>
```

Then you can run the *Record Shell* for each partition:

```
rtirecsh -partition RecordA
rtirecsh -partition RecordB
```

Chapter 6 Using the Replay Tool

Besides replaying data with *Recording Console*, you can use the *Replay* tool directly. You may find this method useful when you want to tie its service into your own infrastructure or software, or if you need to use its more advanced features.

The *Replay* tool replays recorded data by publishing it just like the original *Connext* application did. You can use the original domain ID, QoS settings and data rate, or make changes to test different scenarios.

See also:

- Chapter 7: Configuring the Replay Tool
- Chapter 8: Accessing the Replay Tool from a Remote Location

6.1 Recording Data for Replay

The *Replay* tool can replay information that has been stored in either serialized or deserialized form. If *Replay* is to be used to replay deserialized data, ensure that *all* of the fields of the sample data are recorded, as *Replay* is unable to replay partial data.

Note: SQLite is unable to look at the individual fields in the sample data of files recorded in serialized mode.

6.2 Starting the Replay Tool

Open a command prompt[1] and change to the **<*install-dir*>/scripts** directory. Then enter:

```
> rtireplay -cfgFile <file> -cfgName <configuration>
```

Table 6.1 describes the command-line options and which ones are required.

6.3 Stopping the Replay Tool

To stop the *Replay* tool, use **<Control-c>**.

Table 6.1 **Replay Tool's Command-line Options**

Command-line Option	Description
`-appName <name>`	Specifies an application name which is used to identify the application for remote administration. Default: -cfgName
`-cfgFile <file>`	**Required.** Used to identify the XML configuration file.
`-cfgName <name>`	**Required.** Identifies the configuration within the XML configuration file. The *Replay* tool will load the <replay_service> with the same name as this value.
`-domainIdBase <int>`	Adds this value to the domain IDs in the configuration file. Default: 0
`-forceXmlTypes`	When used with XML Type Configuration, this option instructs *Replay* to always use type code from the XML file, even if an alternate is available from recorded data.
`-help`	Displays this information.
`-identifyExecution`	Appends the host name and process ID to the appName to help using unique names.
`-noAutoEnable`	Use this option if you plan to enable the *Replay* tool remotely.
`-remoteAdministrationDomainId <int>`	Enables remote administration and sets the domain ID for the communication. Default: remote administration is not enabled.
`-srvName <name>`	Specifies a name that will be used to identify the service.

[1] On Windows systems: from the **Start** menu, select **Accessories, Command Prompt**.

Table 6.1 **Replay Tool's Command-line Options**

Command-line Option	Description
`-verbosity <value>`	Specifies what type of logging information should be printed. 0: Silent 1: Exceptions (both *Connext* and the *Replay* tool) 2: Warnings (the *Replay* tool only) 3: Information (the *Replay* tool only) 4: Warnings (both *Connext* and the *Replay* tool) 5: Tracing (the *Replay* tool only) 6: Tracing (both *Connext* and the *Replay* tool) Default: 1
`-version`	Prints the *Replay* tool's version.

6.4 Performance and Indexing

The *Replay* tool replays stored samples in the same order in which they were received, using SQLite indexes to retrieve the samples in sorted order. SQLite automatically builds indexes when opening an SQLite table for sorted access, and for large tables the process of building the index may take some time.

To improve *initialization* performance, the *Replay* tool attempts to create and store indexes (rather than depend upon automatic indexing) for the tables that it will be replaying; this saves initialization time on subsequent replays.

The *Replay* tool*'s* ability to store indexes is controlled by the <readonly> parameter under <replay_database> (see Database (Input File) Properties (Section 7.4)). The default value for <readonly> is false; this allows the *Replay* tool to write the table indices to the database. If you change <readonly> to true, the *Replay* tool will display a message during initialization for each table opened, stating that it was unable to store the table index.

In summary, the *replay* performance of the *Replay* tool is not affected by the <readonly> parameter. The *Replay* tool will use the fastest means of retrieving samples in either case. But setting the <readonly> option to false (the default) may help improve *initialization* performance.

Chapter 7 Configuring the Replay Tool

When you start the *Replay* tool, you must specify a configuration file in XML format. In that file, you can set properties that control the data source, which topics to replay, and attributes such as the replay speed. This chapter describes how to write a configuration file.

7.1 How to Load Replay's XML Configuration File

The *Replay* tool loads its XML configuration from multiple locations. This section presents the various approaches, listed in load order.

The first three locations only contain QoS Profiles and are inherited from *Connext* (see Chapter 15 in the *RTI Core Libraries and Utilities User's Manual*).

❑ **$NDDSHOME/resource/qos_profiles_5.0.x[1]/xml/NDDS_QOS_PROFILES.xml**

This file contains the *Connext* default QoS values; it is loaded automatically if it exists. *(First to be loaded.)*

❑ **File in NDDS_QOS_PROFILES**

The files (or XML strings) separated by semicolons referenced in this environment variable are loaded automatically.

❑ ***<working directory>*/USER_QOS_PROFILES.xml**

This file is loaded automatically if it exists. If the USER_QOS_PROFILES file is found and there is a default profile specified in it, this default profile is automatically applied to the QoS settings of the *Recording Service* entities.

[1] *x* stands for the version letter of the current release.

The next locations are specific to *Replay*:

- ❏ ***<rtiReplay executable location>*/../resource/xml/RTI_REPLAY_SERVICE.xml**

  This file contains the default configuration for the *Replay* tool; it is loaded if it exists. **RTI_REPLAY_SERVICE.xml** defines a configuration that replays all topics on domain 0.

- ❏ ***<working directory>*/USER_REPLAY_SERVICE.xml**

  This file is loaded automatically if it exists.

- ❏ The file specified with the command-line option, **-cfgFile** (see Table 6.1 on page 6-2).

You may use a combination of the above approaches.

7.2 General Format

The Replay configuration file uses XML format. The main sections use the following top-level tags:

Top-level Tag	Reference Section
<replay_service>	General Properties for Replay (Section 7.3)
<replay_database>	Database (Input File) Properties (Section 7.4)
<session>	Session Properties (Section 7.5)
<replay_topic>	Replay Topic Properties (Section 7.6)
<time_control>	Time Control Properties (Section 7.7)
<administration>	Remote Administration Properties (Section 7.8)
<type_config>	Type Configuration (Section 7.9)

The XML configuration file used by *Replay* has a simple hierarchical format. The **replay_service** is configured to replay data contained in one or more **replay_database**. Each **replay_database** is associated with a *DomainParticipant*, and must contain one or more *session*. Each **session** is associated with a *Publisher,* and corresponds to a unique execution thread. Each *session* contains one or more **replay_topic**, each of which is associated with a *DataWriter,* and contains a filter expression that specifies what information contained in the data base should be replayed. Each of the four major levels—**replay_service**, **replay_database**, **session**, and **replay_topic**—may contain a

time_control element that allows control over such features as the rate of replay, how much of the available data to be replayed, and for coordination.

Much of *Replay*'s configuration has been designed to be compatible with the *Record* tool, so familiarity with the *Record* tool's concepts and configuration will be helpful. See Chapter 4: Configuring the Record Tool.

Let's look at a very basic configuration, just to get an idea of its contents. You will learn the meaning of each line as you read the rest of this chapter.

```
<dds>
<replay_service name="default">

    <!-- Optional remote administration configuration -->
    <administration>
        <domain_id> 1 </domain_id>
    </administration>
    <time_control>
        <start_time> 6 </start_time>
    </time_control>
    <auto_exit> yes </auto_exit>

    <replay_database name="simple_config">

        <filename> replay_database.dat </filename>
        <!-- Optional ParticipantQos -->
        <participant>
            <domain_id> 0 </domain_id>
        </participant>

        <session name="simple_session">

            <time_control>
                <rate> 2 </rate>
            </time_control>
            <!-- Optional PublisherQos -->
            <publisher_qos></publisher_qos>

            <replay_topic name="all_topics">

                <time_control>
                    <stop_time> 26 </stop_time>
                </time_control>
                <input>      <!--  Required Values -->
                    <topic_name> * </topic_name>
                    <type_name> * </type_name>
```

```
                            <record_group_name> * </record_group_name>
                            <domain_name> * </domain_name>
                        </input>
                        <!-- Optional Values for writing data -->
                        <output>
                            <!-- Optional DataWriterQos -->
                            <datawriter_qos></datawriter_qos>
                        </output>
                    </replay_topic>
                </session>
            </replay_database>
        </replay_service>
        </dds>
```

7.3 General Properties for Replay

Table 7.1 describes optional properties that control the *Replay* tool's main module.

All <replay_service> properties are optional except replay_database.

These properties must be specified inside **<replay_service name="*String*">** and **</replay_service>** tags, where *String* is the name to be assigned to the service entity when it is created. This name will be used during remote administration unless it is overridden by the <administration> <name> element.

Table 7.1 **Replay Service Properties**

Property	Syntax	Description

administration	`<administration>`
	`Remote Admin. Properties`
	`</administration>`
		Configures the *DomainParticipant* that can be used to remotely control *Replay* via the *rtireplaysh* utility.
		See Remote Administration Properties (Section 7.8).
		The Remote Administration Properties must specify a domain_id. You may also specify a name, participant_qos, publisher_qos, subscriber_qos, datareader_qos, and datawriter_qos.

Table 7.1 **Replay Service Properties**

Property	Syntax	Description

auto_exit	`<auto_exit>`
	`  DDS_Boolean`
	`</auto_exit>`
		Controls whether or not the *Replay* tool should terminate when all the available data specified in the initial configuration has been replayed.
		Default: False

replay_database	`<replay_database>`
	`  Replay Database Properties`
	`</replay_database>`
		Required. Specifies configuration properties that describe how to replay the information from a database. This element can be repeated.
		See Database (Input File) Properties (Section 7.4)

time_control	`<time_control>`
	`  Time Control Properties`
	`</time_control>`
		Specifies time configuration properties to be applied to the *Replay* tool as a whole.
		See Time Control Properties (Section 7.7).

7.4 Database (Input File) Properties

Table 7.2 describes the source of the data that *Replay* will replay.

All <replay_database> properties are optional except session.

These properties must be specified inside **<replay_database name="*String*">** and **</replay_database>** tags, where *String* is the name to be assigned to the database entity when it is created. This name will be used during remote administration.

Table 7.2 **Replay Database Properties**

Properties under <replay_ database>	Syntax	Description

filename	`<filename>`
	`  String`
	`</filename>`
		Specifies the name of the fileset that contains the data to be replayed.
		Default: Undefined

participant	`<participant>`
	`  Participant Properties`
	`</participant>`
		See Table 7.3, "Participant Properties"

Table 7.2 **Replay Database Properties**

Properties under <replay_ database>	Syntax	Description

readonly	`<readonly>`
	`DDS_Boolean`
	`</readonly>`
		Specifies if *Replay* should open the data file in read-only mode (true), or read-write mode (false). Setting this option to false is useful to enable indexing of older database files.
		Default: False
		See Performance and Indexing (Section 6.4).

session	`<session>`
	`Session Properties`
	`</session>`
		Required. The configuration properties that describe how to replay the information in a session. This element can be repeated.
		See Table 7.4, "Session Properties"

time_control	`<time_control>`
	`Time Control Properties`
	`</time_control>`
		The time configuration properties to be applied to the replay database.

type_config	`<type_config>`
	`XML Properties`
	`</type_config>`
		Optional XML type configuration for this *replay_database*. This option is useful when type codes have not been recorded in the database, or when specifying types that are too large to be recorded in the database.
		See Table 7.13, "XML Type Configuration Properties"

Table 7.3 **Participant Properties**

Properties under <participant>	Syntax	Description

domain_id	`<domain_id>`
	`DDS_Long`
	`</domain_id>`
		Sets the domain ID. Default: 0

participant_qos	`<participant_qos>`
	`DDS_QosPolicy`
	`</participant_qos>`
		Configures the DomainParticipant's QoS policies. See the *RTI Core Libraries and Utilities User's Manual's* chapter on Configuring QoS with XML.
		Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on DomainParticipants.

7.4.1 Enabling Monitoring Library with Replay

This section only applies if you want to use *RTI Monitoring Library*, a separate *RTI Connext* component that enables *Connext* applications to provide monitoring data. The monitoring data can be visualized with *RTI Monitor*, a separate GUI application that can run on the same host as *Monitoring Library* or on a different host. *Recording Service* is statically linked to *Monitoring Library* (you do not have to install it separately).

To enable monitoring in the *Replay* tool, use the same approach described in Enabling Monitoring Library in the Record Tool (Section 4.5.1). In the <replay_database> section, include the **rti.monitor.library** property with the value **rtimonitoring**. For example:

```
<participant>
    <domain_id>0</domain_id>
    <participant_qos>
        <property>
            <value>
                <element>
                    <name>rti.monitor.library</name>
                    <value>rtimonitoring</value>
                    <propagate>false</propagate>
                </element>
            </value>
        </property>
    </participant_qos>
</participant>
```

7.5 Session Properties

Table 7.4 describes the Session's properties.

All <session> properties are optional except replay_topic.

These properties must be specified inside **<session name="*String*">** and **</session>** tags, where *String* is the name to be assigned to the session entity when it is created. This name will be used during remote administration.

7.6 Replay Topic Properties

Table 7.5 describes the Topics' properties.

Table 7.4 **Session Properties**

Properties under <session>	Syntax	Description
publisher_qos	<publisher_qos> *DDS_QosPolicy* </publisher_qos>	Configures the Publisher's QoS policies. See the *RTI Core Libraries and Utilities User's Manual's* chapter on Configuring QoS with XML. Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on Publishers.
replay_topic	<replay_topic> *Replay Topic Properties* </replay_topic>	**Required.** The configuration properties that describes the topics to be replayed, and the associated *DataWriter* configuration. This element can be repeated. See Table 7.5, "Replay Topic Properties"
thread	<thread> *Thread Properties* </thread>	Configures the properties for the execution thread.
time_control	<time_control> *Time Control Properties* </time_control>	Configures the time control properties to be applied to the Session.

All <replay_topic> properties are optional except input.

These properties must be specified within **<replay_topic name="*String*">** and **</replay_topic>** tags, where *String* is the name to be assigned to the replay topic entity when it is created. This name will be used during remote administration.

All input properties (Table 7.6) are required, except for type_name, which is optional.

All output properties (Table 7.7) are optional.

Table 7.5 **Replay Topic Properties**

Properties under <replay_topic>	Syntax	Description
input	<input> *Input Properties* </input>	**Required.** Configures the topics that are to be replayed from the database. See Table 7.6, "Input Properties".

Table 7.5 **Replay Topic Properties**

Properties under <replay_topic>	Syntax	Description

output	`<output>`
	`Output Properties`
	`</output>`
		Configures the attributes to be used in writing the replayed topics. See Table 7.7, “Output Properties”.

time_control	`<time_control>`
	`Time Control Properties`
	`</time_control>`
		Specifies time configuration properties to be applied to the Session.

Table 7.6 **Input Properties**

Properties under <input>	Syntax	Description

domain_name	`<domain_name>`
	`String`
	`</domain_name>`
		Required. Specifies the name of the *domain_name* that was specified in the *Record* tool's configuration file or a regular or wildcard expression.

record_group_name	`<record_group_name>`
	`String`
	`</record_group_name>`
		Required. Specifies the name of the *record_group* that was specified in the *Record* tool's configuration file or a regular or wildcard expression.

topic_name	`<topic_name>`
	`String`
	`</topic_name>`
		Required. Specifies the name of the *topic_name* that was specified in the *Record* tool's configuration file or a regular or wildcard expression.

type_name	`<type_name>`
	`String`
	`</type_name>`
		Specifies the name of the type_code to be used in writing matching topics. This parameter will default to "*" if not specified. *Replay* will search for a matching type name only within matching topic records.

7.7 Time Control Properties

The <time_control> element can be applied to any of *Replay's* major entities: <replay_service>, <replay_database>, <session>, and <replay_topic>.

Table 7.7 **Output Properties**

Properties under <output>	Syntax	Description

datawriter_ qos	`<datawriter_qos>`
	`DDS_DataWriterQos`
	`</datawriter_qos>`
		Specifies the QoS settings for all *DataWriters* created for this Replay_Topic. A *DataWriter* is created for each *Topic* that matches the **topic_expr**. All the *DataWriters* for the Replay_Topic will use the same set of QoS policies. You can specify all of the QoS policies with this datawriter_qos property.
		See the *RTI Core Libraries and Utilities User's Manual's* chapter on Configuring QoS with XML.

topic_name	`<topic_name>`
	`String`
	`</topic_name>`
		Specifies the name to be assigned to the topic when creating a *DataWriter* to write the data to be replayed.

topic_qos	`<topic_qos>`
	`DDS_TopicQos`
	`</topic_qos>`
		Specifies the QoS settings to be applied to the topic when creating a *DataWriter* to write the data to be replayed.

type_name	`<topic_name>`
	`String`
	`</topic_name>`
		Specifies the name to be assigned to the type when creating a *DataWriter* to write the data to be replayed.

❑ The *index time* of the <replay_service> is the earliest *index time* of all of its component **replay_database** entities.

❑ The *index time* of a <replay_database> is the earliest timestamp of the database, taken from its creation log.

❑ The *index time* of a <replay_topic> is the earliest timestamp of the topic, taken from the first recorded sample of the topic.

All time control properties are optional.

The **start_time** and **stop_time** values of a child entity are constrained by the **start_time** and **stop_time** settings of its parent entities. If a **start_time** or **stop_time** value is explicitly specified and constrained by one of its parent entities, *Replay* will issue a warning that the value has been truncated.

The **start_mode** of a child entity overrides the **start_mode** setting of its parent entities.

The **time_mode** value of a child entity cannot be applied to a parent entity (e.g., TOPIC_RELATIVE cannot be applied to the **time_control time_mode** element of a <session>, <replay_database>, or <replay_service>).

Table 7.8 **Time Control Properties**

Properties under <time_ control>	Syntax	Description

rate	`<rate>`
	`Real Number`
	`</rate>`
		Specifies the replay rate, expressed as a multiple of the original rate at which the data was recorded. (Therefore 1 means the same as the original rate, 2 means twice as fast, etc.)
		Although this rate may be specified as a real decimal number, the internal resolution of the rate value is stored as a percentage with two decimal places. The rate may also be configured with the special value "AS_FAST_AS_POSSIBLE", which directs *Replay* to replay the data without any intervening time between samples.
		The minimum value is 0.01 (1% of the original rate.)
		Default: 1

start_mode	`<start_mode>`
	`Start Mode`
	`</start_mode>`
		Sets the starting mode of the entity, as described in Table 7.9, "Start Mode Values". Default: **AUTOMATIC**

start_offset	`<start_offset>`
	`DDS_Duration`
	`</start_offset>`
		The time to offset the selected entity's starting time from its parent entity. This value is used for synchronizing data that is replayed from different sources.
		When applied to the <replay_service>, *Replay* will delay for the number of seconds specified between the creation of the entities and the start of replay. (To allow for discovery, for example).
		When applied to a <replay_database>, this time is the amount of offset between the *index time* of the replay_service and the *index time* of the database.
		When applied to a <replay_database>, this time allows an additional *DDS_Boolean* element, **<auto_offset>**, which if set *TRUE* directs *Replay* to automatically calculate the offset between the *index time* of the <replay_database> and the *index time* of the <replay_service>. This value should not be applied to <session> or <replay_topic>.
		Default: 0

start_time	`<start_time>`
	`DDS_Duration`
	`</start_time>`
		The time of the recorded data at which replay is to begin. The time is interpreted based on the setting of time_mode**.**
		Although expressed as <sec> and <nanosec>, the internal resolution of *Replay* is limited to milliseconds. Default: 0

Table 7.8 **Time Control Properties**

Properties under <time_ control>	Syntax	Description

stop_time	`<stop_time>`
	`  DDS_Duration`
	`</stop_time>`
		The time of the recorded data at which replay is to stop. The time is interpreted based on the setting of time_mode**.**
		Although expressed as <sec> and <nanosec>, the internal resolution of *Replay* is limited to milliseconds.
		Default: Infinity

time_mode	`<time_mode>`
	`  Time Mode`
	`</time_mode>`
		Describes how the **start_time** and **stop_time** parameters should be interpreted. See Table 7.10, "Time Mode Values".
		Default: **DATABASE_RELATIVE**, except when applied to a **replay_service** entity, whose default is **SERVICE_RELATIVE**.

Table 7.9 **Start Mode Values**

Enumeration Value	Description
AUTOMATIC	Replay of the entity begins automatically. For subordinate entities, replay begins when parent replay starts.
LOOP	Replay of the selected section begins automatically, and is restarted immediately after the last data sample of the *entity* has been replayed. For example, **replay_topics** with **start_mode LOOP** will each restart as soon as each topic has completing its replay, while a **session** with **start_mode LOOP** will restart only when *all* of its topics have completed replay. **Note:** Currently this mode is operational only for **session** and **replay_topic** entities.
MANUAL	Replay of the entity begins when explicitly directed by remote administration. When an entity is manually started, all of its child entities with **AUTOMATIC start_mode** will also be started at the same time, and so forth continuing to the lowest child.
MATCHED	*CURRENTLY NOT SUPPORTED.* *Replay begins after each child DataWriter has detected at least one matched reader.*

Table 7.10 **Time Mode Values**

Enumeration Value	Description
ABSOLUTE	The **start_time** and **stop_time** values are in absolute timestamps and will be used without modification.
DATABASE_ RELATIVE	The **start_time** and **stop_time** values are relative to the **replay_database**'s *index time* (i.e., the *index time* of the **replay_database** will be added to the **start_time** and **stop_time** values, if specified).
SERVICE_ RELATIVE	The **start_time** and **stop_time** values are relative to the **replay_service**'s *index time* (i.e., the *index time* of the **replay_service** will be added to the **start_time** and **stop_time** values, if specified).
TOPIC_ RELATIVE	*CURRENTLY NOT SUPPORTED.* The **start_time** and **stop_time** values are relative to the earliest timestamp of the topic (i.e., the *index time* of the topic will be added to the **start_time** and **stop_time** values, if specified).

7.8 Remote Administration Properties

The *Replay* tool can be controlled remotely, by either the *rtireplaysh* utility, or by a *Connext* application that reads and writes the remote administration topic.

For security reasons, Remote Administration is turned off in the *Replay* tool by default.

The Remote Administration section of the configuration file is used to enable Remote Administration and configure its behavior. This section is not required in the configuration file.

The *rtireplaysh* utility or a remote application can send commands to the *Replay* tool to:

- ❑ Start/stop/pause/resume/step the *Replay* tool or any of its individual entities.
- ❑ Change the speed of replay of the *Replay* tool or any of its individual entities.
- ❑ Query the status of the *Replay* tool or any of its individual entities.
- ❑ Reposition the *Replay* tool or any of its entities to any point in the replay range specified for playback.

Performance and Indexing (Section 6.4) describes the command format and individual commands available in *rtireplaysh, Replay's* remote administration utility.

Table 7.11 describes the Remote Administration properties. All remote administration properties are optional.

All Remote Administration properties must be specified inside **<administration>** and **</administration>** tags.

Table 7.11 **Remote Administration Properties**

Properties under <administration>	Syntax	Description

datareader_qos	`<datareader_qos>`
	`DDS_DataReaderQos`
	`</datareader_qos>`
		Configures the QoS for the DataReader created by the *Replay* tool's Remote Access module.
		Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on DataReaders.

datawriter_qos	`<datawriter_qos>`
	`DDS_DataWriterQos`
	`</datawriter_qos>`
		Configures the QoS for the DataWriter created by the *Replay* tool's Remote Access module.
		Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on DataWriters.

distributed_ logger	`<distributed_logger>`
	`Distributed Logger Properties`
	`</distributed_logger>`
		Configures *RTI Distributed Logger*.
		See Enabling RTI Distributed Logger in the Replay Tool (Section 7.8.1).

domain_id	`<domain_id>`
	`DDS_Long`
	`</domain_id>`
		Sets the domain ID.
		Default: 0

name	`<name>`
	`String`
	`</name>`
		Assigns a name to the replay service. You can use this name when sending commands via *Replay Shell* (see Section 6.4).

participant_qos	`<participant_qos>`
	`DDS_ParticipantQos`
	`</participant_qos>`
		Configures the QoS for the Participant created by the *Replay* tool's Remote Access module.
		Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on DomainParticipants.

publisher_qos	`<publisher_qos>`
	`DDS_PublisherQos`
	`</publisher_qos>`
		Configures the QoS for the Publisher created by the *Replay* tool's Remote Access module.
		Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on Publishers.

Table 7.11 **Remote Administration Properties**

Properties under <administration>	Syntax	Description

status_period	`<status_period>`
	`DDS_Duration`
	`</status_period>`
		Specifies, in seconds and nanoseconds, the period between each status message sent by the Replay Service to the Replay Shell.
		When this value is set to zero (the default), no status message is sent.
		Applications that want to periodically poll the status of the Replay service they administer should provide a value for this property.
		Default: 0 (no status is sent)

subscriber_qos	`<subscriber_qos>`
	`DDS_QosPolicy`
	`</subscriber_qos>`
		Configures the QoS for the Subscriber created by the *Replay* tool's Remote Access module.
		Defaults: See the *RTI Core Libraries and Utilities* API Reference HTML documentation on Subscribers.

7.8.1 Enabling RTI Distributed Logger in the Replay Tool

The *Replay* tool provides integrated support for *RTI Distributed Logger*.

When you enable *Distributed Logger,* the *Replay* tool will publish its log messages to *Connext*. Then you can use *RTI Monitor*[1] to visualize the log message data. Since the data is provided in a *Connext* topic, you can also use *rtiddsspy* or even write your own visualization tool.

To enable *Distributed Logger*, modify the *Replay* tool's XML configuration file. In the <administration> section, add the <distributed_logger> tag as shown in the example below.

```
<administration>
    <name>Replay Service using Distributed Logger</name>
    <domain_id>99</domain_id>
    <distributed_logger>
        <enabled>true</enabled>
    </distributed_logger>
```

[1] *RTI Monitor* is a separate CUI application that can run on the same host as your application or on a different host.

```
    </administration>
```

There are more configuration tags that you can use to control *Distributed Logger's* behavior. For example, you can specify a filter so that only certain types of log messages are published. For details, see the *RTI Distributed Logger Getting Started Guide.*

7.9 Type Configuration

The <type_config> element allows you to pass type configuration information to the *Replay* tool in the form of XML type-configuration files.

Table 7.12 describes the Type Configuration properties. All Type Config properties are optional.

Table 7.12 **Type Configuration Properties**

Properties under <type_config>	Syntax	Description

xml	`<xml>`
	`   XML Type Configuration Properties`
	`</xml>`
		Allows you to specify XML type-configuration files, the path in which to find them, and other properties related to the registration of the types with the DomainParticipants.
		The XML type configuration for this domain group. See Table 7.13, "XML Type Configuration Properties".

Table 7.13 **XML Type Configuration Properties**

Properties under <xml>	Syntax	Description

file_group	`<file_group>`
	`  <element>`
	`    File Group Properties`
	`  </element>`
	`<file_group>`
		Allows you to specify XML file groups; each of these files contain type definitions to be used by the Replay tool. The element tag can be repeated.
		See Table 7.14, "File Group Properties".

Table 7.13 **XML Type Configuration Properties**

Properties under <xml>	Syntax	Description

max_sequence	`<max_sequence>`
	`Integer`
	`</max_sequence>`
		The default sequence size, in case there are unbounded sequences in the type definitions specified in the any of the files specified in any of the file groups.

max_string	`<max_string>`
	`Integer`
	`</max_string>`
		The default string size, in case there are unbounded strings in the type definitions specified in any of the files specified in any of the file groups.

path	`<path>`
	`<element>`
	`Path`
	`</element>`
	`</path>`
		A list of the paths (relative or absolute) to be used when searching for the XML type definition files. The <element> tag can be repeated.

register_top_level	`<register_top_level>`
	`Boolean`
	`</register_top_level>`
		Whether or not to register the top-level types found in the type definitions with their canonical names. Do this with any of the files defined in any of the file groups. Default: TRUE.

Table 7.14 **File Group Properties**

Properties under <file_group> <element>	Syntax	Description

file_name	`<file_name>`
	`<element>`
	`File Name string`
	`</element>`
	`</file_name>`
		A list of file name strings, specifying files containing the type definitions. The element tag can be repeated.

type	`<type>`
	`<element>`
	`Type Registration Properties`
	`</element>`
	`</type>`
		Specific type properties. A list of type registration properties used to define how a type found in the files has to be registered by Recorder in the DomainParticipants. See Table 7.15, "Type Registration Properties"

Table 7.14 **File Group Properties**

Properties under <file_group> <element>	Syntax	Description

max_sequence	`<max_sequence>`
	`Integer`
	`</max_sequence>`
		The default sequence size in case there are unbounded sequences in the type definitions specified in the any of the files specified in this specific file group.

max_string	`<max_string>`
	`Integer`
	`</max_string>`
		The default string size in case there are unbounded strings in the type definitions specified in any of the files specified in this specific file group .

register_top_level	`<register_top_level>`
	`Boolean`
	`</register_top_level>`
		Whether or not to register the top-level types found in the type definitions with their canonical names. Do this with any of the files found in this specific file group. The value of this setting overrides the value of the upper-level setting (see Table 7.13, "XML Type Configuration Properties"). Default: TRUE.

Table 7.15 **Type Registration Properties**

Properties under <type> <element>	Syntax	Description

type_name	`<type_name>`
	`Type Name string`
	`</type_name>`
		A string representing the canonical type name as defined in the XML type definition for the type.

registered_ type_name	`<registered_type_name>`
	`<element>`
	`Registration name string`
	`</element>`
	`</registered_type_name>`
		When registering the type with the DomainParticipant, this setting defines a list of names to register the type with. The element tag can be repeated.

Table 7.15 **Type Registration Properties**

Properties under <type> <element>	Syntax	Description

register_top_level	`<register_top_level>`
	`    Boolean`
	`</register_top_level>`
		Whether or not to register this type's canonical name (as defined in tag type_name) with the DomainParticipant. This name is registered alongside any of the registration type names defined in tag registered_type_name above. Default: TRUE.

7.10 Recording Service Integration with Extensible Types

The *Replay* tool includes partial support for the "Extensible and Dynamic Topic Types for DDS" specification from the Object Management Group (OMG)[1]. This section assumes that you are familiar with Extensible Types and you have read the *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types.*

- ❑ *Replay* can publish topics associated with final and extensible types.
- ❑ Users can provide an XML definition of the type that must be used by *Replay* to publish a topic. The XML description supports structure inheritance (described in the *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types*).

7.10.1 Providing XML Type Description

With Extensible Types, it is possible to have multiple versions of a type associated with a topic “T”. *Recording Service* allows you to select the type version that will be used to record the data (see Section 4.10). However, *Recording Service* persists the type description (TypeCode) corresponding to each one of the discovered versions in the DCPSPublication table.

This may lead to type conflicts in the *Replay* tool, since when the service is configured to replay topic “T”, it will pick the first type version that was discovered—even if that version does not correspond to the version used to record the data.

[1] http://www.omg.org/spec/DDS-XTypes/

To resolve this problem, you can provide the XML definition for topic “T” using the **<type_config>** tag under **<xml>**. In addition, you will have to run the *Replay* tool with the command line option **-forceXMLTypes** to force it to use the XML definition.

For additional information on how to provide XML types to the *Replay* tool, see Section 7.9.

Chapter 8 Accessing the Replay Tool from a Remote Location

The *Replay Shell* is a *Connext* application that can remotely control the *Replay* tool.

To start the Replay Shell:

Open a command prompt[1] and change to the ***<install-dir>*/scripts** directory. Then enter:

```
> <install dir>/scripts/rtireplaysh [options]
```

Table 8.1 lists the command-line options you can use when starting the *Replay Shell*. Once it is started, you can use the commands in Table 8.2.

Table 8.1 **Replay Shell's Command-Line Options**

Command-line Option	Description
`-cmdFile <file>`	A file that contains commands to be run.
`-domainId <integer>`	Specifies the domain ID, an integer between 0 and 232. Default: 0
`-help`	Prints version information and a list of options.
`-timeout <seconds>`	Maximum number of seconds to wait for a remote response. Default: 15 seconds

[1] On Windows systems: from the **Start** menu, select **Accessories, Command Prompt**.

Table 8.1 **Replay Shell's Command-Line Options**

Command-line Option	Description
`-verbosity <value>`	Specifies what type of logging information should be printed. 0: Silent 1: Exceptions (both *Connext* and the *Replay* tool) 2: Warnings (the *Replay* tool only) 3: Information (the *Replay* tool only) 4: Warnings (both *Connext* and the *Replay* tool) 5: Tracing (the *Replay* tool only) 6: Tracing (both *Connext* and the *Replay* tool) Default: 1

The *Replay Shell* commands use this format:

```
<command> <replay_service> [entity] [value]
```

where:

- ❑ **<command>** is one of the supported commands (see Table 8.2).
- ❑ **<replay_service>** is the name given to the *Replay* service by one of the following, in descending order of precedence:
  - The value specified with the **-appName** command-line option used when starting the *Replay* tool *(highest precedence)*
  - The value for the <replay_service><administration><name> element (see page 7-14)
  - The value for the <replay_service> name attribute *(lowest precedence)* (see Section 7.3)
- ❑ **[entity]** is any one of the service entities expressed in this hierarchical form: <database-name>[::<session-name>[::<topic-name>]].

  Note: *In this release, not all commands are supported for all entity levels.* Please see the *Recording Service* Release Notes for details on which modes are currently supported.

  The database-name must match a name from a <replay_database> tag in the configuration file that you specified when starting the *Replay* tool , such as:

  ```
  <replay_database name="simple_config">
  ```

  Similarly, if you specify a session-name, it must match a name from a <session> tag within the specified database, such as:

  ```
  <session name="A_Session">
  ```

If you specify a topic-name, it must match a name from a <replay_topic> tag within the specified session, such as:

```
<replay_topic name="All_Topic">
```

If you do not specify an entity, the command is applied to the replay service itself.

❑ **value** depends on the command, see Table 8.2. Not all commands require a value.

Table 8.2 **Replay Shell's Commands**

Command	Description
`exit`	Exits the shell.
`goto`	Repositions an entity to a specific point in the playback range (relative to the entity's start and end times). This command takes a timestamp argument, which is a string of digits of the form "SSSSSSSSSSUUUUUU". The first ten digits specify seconds and the last six digits specify microseconds.
`pause`	Pauses replay of an entity.
`query`	Returns the status of an entity, including: • If the entity is enabled, started, pending, paused, and completed • The number of child topics owned by the entity • The number of active child topics owned by the entity The format of this status is "[*ccccc dd dd*]", where each *c* is either 'T' or 'F', and 'dd' is a decimal number. The 'T' and 'F' entries represent enabled, started, pending, paused, and completed. The *d*'s are decimal numbers for how many child topics are owned by the entity and how many active child topics are owned by the entity.
`rate`	Changes the replay rate of an entity. The rate is a multiplier from 0.1 to 4 billion. It replays at the speed of the multiplier (2 = 2x, 0.5 = 1/2x, etc.) Default: 1
`resume`	Resume replay of an entity.
`start`	Starts replay of an entity.
`step`	Replays a single sample from the entity.
`stop`	Stops replay of an entity.

Chapter 9 Viewing Recorded Data with SQLite

The *Record* tool stores data in a SQL database. This chapter describes how the data is stored and how to view the data with the provided SQL command-line tool, **sqlite3**.

Important: For information on SQL commands, please visit www.sqlite.org.

To open a recorded file, start **sqlite3**. For example:

```
$ cd rtirecord.5.0.x-ndds.5.0.y/bin/<architecture>
$ sqlite3 <recorded file>
```

Then you can list all the available topics by entering:

```
sqlite> .tables
```

This will list the tables (one per topic) in the database file. For example:

```
Circle$MyGroup$MyDomain
DCPSParticipant
DCPSPublication
DCPSSubscription
RTILog
RTIVersion
Square$RecordAll$MyDomain
Square$RecordXYSquaresinMyDomain$MyDomain
Triangle$RecordAll$MyDomain
```

You can query the tables using standard SQL syntax. For example, assume that the Topic Circle was recorded in RecordGroup, MyGroup, in domain MyDomain. In this case, the *Record* tool creates a table called Circle$MyGroup$MyDomain to store all data published on Topic Circle. To list all data received on Topic Circle, enter:

```
sqlite> select * from Circle$MyGroup$MyDomain;
```

Note: For more example commands, please see the Chapter 4 in the Getting Started Guide.

To exit **sqlite3**, enter:

```
sqlite> .exit
```

9.1 Format of the Recorded Data

9.1.1 Discovery Data

The *Record* tool stores discovery-related data in these tables:

❑ DCPSParticipant — corresponds to the Participant Built-in Topic

❑ DCPSPublication — corresponds to the Publication Built-in Topic

❑ DCPSSubscription — corresponds to the Subscription Built-in Topic

Please refer to the *RTI Core Libraries and Utilities* C API documentation for the fields in each of the corresponding builtin topics. (In the HTML documentation for the C API, select **Modules, Domain Module, Built-in Topics**.)

The *Record* tool stores the following information, in this order:

❑ A timestamp (in microseconds since Jan. 1st, 1970).

❑ State (internal information, can be ignored)

❑ Type (internal information, can be ignored)

❑ The domain ID from which the sample was received.

❑ The BuiltinTopicData.

❑ The SampleInfo information, as described in the *Connext* documentation.

Note: When using self-contained database files, the **locator_filter** column of the Publication Built-in Topic Data will not be replicated. The Subscription Built-in Topic Data will also not be replicated. This is done to minimize the overhead of replication when opening a new database file.

9.1.2 User Data

❏ Deserialized Data

When the *Record* tool stores data in deserialized form, it creates a mapping from a Topic to a table. Each individual scalar is stored in a column named with the fully qualified name.

For example, the following will create a column, **bar$x**:

```
struct Bar {
    long x;
};

struct Foo {
    Bar bar;
};
```

❏ Topic Data

The *Record* tool creates a table called "TopicName$RecordGroupName$DomainName" for each recorded topic (unless the shared_table property is true).

The *Record* tool stores topic data as specified in the subscription properties.

For each topic, the *Record* tool also stores the following, in this order:

- A timestamp (in microseconds since Jan. 1st, 1970). (Note that the column name is ___timestamp, with 3 underscores.) This is the time that the data was committed to the database.
- The domain ID from which the sample was received.
- Table_prefix - RecordGroupName$DomainName (only if <shared_table> is specified)
- Whether the data is stored in serialized or deserialized format.
- The SampleInfo information, as described in the *Connext* documentation.
- Topic data.

If the topic data is saved in serialized form, a special table is used with the following columns:

- serialized_sample: raw data
- serialized_length: length of the raw data
- serialized_endian: 1 — little endian, 0 — big endian

9.1.3 Other Tables

You will notice that the *Record* tool creates two additional tables, RTILog and RTIVersion. You do not need to use these tables, they are for internal use by the *Record* tool.

Chapter 10 Exporting Recorded Data

Recording Service includes a conversion utility that enables serialized or deserialized data recorded with the *Record* tool to be exported to CSV, HTML, SQL, or XML formats. The utility merges the data from all the segments in a fileset. It converts recorded data (either serialized or deserialized) into one of the available formats. The output data is deserialized.

The executable, **rtirecconv**, is located in **<*install dir*>/bin/<*architecture*>**. The script to launch the *Convert* tool is in **<*install dir*>/scripts/rtirecconv**.

To see a list of available options, enter:

```
$ rtirecconv -help
```

You will see the following:

```
rtirecconv [options] fileset|filename
Options:
-help             - Print out this text
-version          - Print out the version
-verbosity [0..6] - verbosity from 0 to 6
-forceXMLTypes    - Always use XML type definitions, if available
-compact <mode>   - How to read octet/char arrays and sequences. Default: auto
      auto - Detect automatically how data is stored and compact/do not compact
             each type of data accordingly.
      yes  - Show the structures with compact data regardless of how they are
             stored.
      no  -  Do not compact the structures regardless of how they are stored.
-tableExpr <expr> - POSIX fn-name expressions, Can be repeated
-includeInfo      - Include the DDS_SampleInfo structure for each sample
-includeDiscovery - Include discovery traffic
-includeNonData   - Include non-data samples
-decodeUnknown    - Try to decode samples from DDS_DataWriters with no typecode
-format <format>  - Output format. Default: xml
      xml  - Output in XML format
      csv  - Output CSV format
      html - Output in HTML table format
      sql  - Output in SQL table format
-decodeChar <format> - Decode char data in this format. Default: text
      hex  - Decode char data as hex
      text - Decode char data as ASCII text
-decodeOctet <format> - Decode octet data in this forma. Default: hex
      hex  - Decode octet data as hex
      text - Decode octet data as ASCII text
-time <format>  - Format for timestamp
      epoch - Show timestamp in microsec from January 1 1970
      gmt   - Show timestamp in GMT
-filePrefix <prefix> - Create one file per table called <prefix>_<Table>
-outputFile <file> - Name of output file, cannot be used with file_prefix
-typeConfig <file> <cfgName> - Name and configuration of the type-configuration
           file
```

> **Note: -typeConfig <file> <cfgName>** specifies the file and configuration name of the *Recording Service* XML configuration, including the type definitions that will be used to convert the data. You should only use this option when you want to provide type definitions using XML. To force the usage of XML type definitions, you will also need to use **-forceXMLTypes**.

For example, assume that recorded data is stored in a file named **mydata.dat_0**. The command to convert the recorded data into XML format is:

```
$ rtirecconv mydata.dat_0
```

An output file called **mydata.dat_0.xml** will be created.

To convert to an HTML table instead, the command is:

```
$ rtirecconv -format html mdata.dat_0
```

This will create an output file called **mydata.dat_0.html**.

Note: The *Convert* tool cannot process partially recorded deserialized data. If you want to record only a subset of the fields in the data structure, you should record in *serialized* format. For deserialized recorded data, it is better to use sqlite to output the data instead of *rtirecconv.*

10.1 Integration with Extensible Types

The *Converter* utility includes partial support for the "Extensible and Dynamic Topic Types for DDS" specification from the Object Management Group (OMG)[1]. This section assumes that you are familiar with Extensible Types and you have read the *Core Libraries and Utilities Getting Started Guide Addendum for Extensible Types*.

- ❑ *Converter* can export final and extensible types.
- ❑ You can provide an XML definition of the type that is used to export the data for a Topic. This will be needed in two cases:
  - *Recording Service* does not provide the type definition in the DCPSPublication table.
  - *Recording Service* provides multiple type-definitions for the Topic in the DCPSPublication table (see Section 10.1.1).

10.1.1 Providing XML Type Description

With Extensible Types, it is possible to have multiple versions of a type associated with a topic “T”. *Recording Service* allows you to select the type version that will be used to record the data (see Section 4.10). However, *Recording Service* persists the type description (TypeCode) corresponding to each one of the discovered versions in the DCPSPublication table.

This may lead to type conflicts in *Converter,* since when the utility converts topic “T”, it will pick the first type version that was discovered for the Topic—even if that version does not correspond to the version used to record the data.

To resolve this problem, you can provide the XML definition for topic “T” using the command line option **-typeConfig**. In addition, you will have to run *Converter* with the command line option **-forceXMLTypes** to force it to use the XML definition.

Converter uses the XML configuration file of *Recording Service* to get the type definitions. For additional information on how to provide XML types to *Recording Service*, see Section 4.9 and then example in Section 4.10.

[1] http://www.omg.org/spec/DDS-XTypes/

Chapter 11 Example Configuration Files

This chapter shows how to configure the *Record* tool for a variety of situations:

- ❑ How to Record All Topics in a Single Domain (Section 11.1)
- ❑ How To Record a Subset of Data from Multiple Domains (Section 11.2)
- ❑ How To Record Data to Multiple Files (Section 11.3)
- ❑ How To Record Serialized Data (Section 11.4)
- ❑ How To Record Using Best-Effort Reliability (Section 11.5)
- ❑ How To Enable Remote Access (Section 11.6)

11.1 How to Record All Topics in a Single Domain

Scenario

You have a system with several nodes using domain ID 54. You want all the data in this system to be recorded to a single file called **mydomaindata**. When the file is full, recording should stop. The typecodes are available from the system.

Configuration File

```
<dds>
<recorder name=”scenario1”>
    <record_database>
        <database_name> mydomaindata </database_name>
        <max_file_size> 1 GB </max_file_size>
    </record_database>
    <domain name="mydomain">
        <domain_id> 54 </domain_id>
    </domain>
    <topic_group name="All">
        <topics>
           <topic_expr> * </topic_expr>
        </topics>
        <field_expr> * </field_expr>
    </topic_group>
    <record_group name="sub0">
        <domain_ref>
           <element> mydomain </element>
        </domain_ref>
        <topic_ref>
           </element> All </element>
        </topic_ref>
    </record_group>
</recorder>
</dds>
```

Expected Outcome

The expected outcome is a single file about 4 GB with all the data in a file called **mydomaindata_0_0**. By default, the *Record* tool will store deserialized data, so the file will have one column for each field in the topic.

11.2 How To Record a Subset of Data from Multiple Domains

Scenario

You have a system with multiple domains, including domain IDs 54 and 98, and hundreds of topics whose names contain “Sensor” (such as TemperatureSensor, HeatSensor, SensorTypes, etc.), in addition to hundreds of other topics. You only want to record the topics that start with “Sensor”, and from each of these topics, you only want to record the fields whose name includes “value” (such as value_max, value_min, current_value).

Configuration File

```
<dds>
<recorder name=”scenario2”>
    <record_database>
        <database_name> mydomaindata </database_name>
        <max_file_size> 1 GB </max_file_size>
    </record_database>
    <domain name="mydomain54">
        <domain_id> 54 </domain_id>
        </domain>
        <domain name="mydomain98">
            <domain_id> 98 </domain_id>
        </domain>
        <topic_group name="Sensor">
            <topics>
               <topic_expr> Sensor* </topic_expr>
            </topics>
            <field_expr> *value* </field_expr>
        </topic_group>
        <record_group name="sub0">
            <domain_ref>
               <element> mydomain54 </element>
            </domain_ref>
            <topic_ref>
               </element> All </element>
            </topic_ref>
        </record_group>
        <record_group name="sub1">
            <domain_ref>
               <element> mydomain98 </element>
            </domain_ref>
            <topic_ref>
               </element> All </element>
            </topic_ref>
        </record_group>
    </recorder>
    </dds>
```

Expected Outcome

The expected outcome is a single file about 4 GB, with all the data in a file called mydomaindata_0_0. By default, the *Record* tool will store deserialized data; so the file will have one column per field in the topic.

11.3 How To Record Data to Multiple Files

Scenario

The *Record* tool is recording data on a system that supports files up to 4 GB in size. However, you want to record more than 4 GB of data.

Configuration File

```
<dds>
<recorder name=”scenario3”>
    <record_database>
        <database_name> mydomaindata </database_name>
        <max_file_size> 2000 kB </max_file_size>
        <max_file_segments> 1000 </max_file_segments>
        <rollover> yes </rollover>
    </record_database>
...
</recorder>
</dds>
```

Expected Outcome

Up to 1,000 files will be created if necessary, named **mydomaindata_0_0**, **mydomaindata_0_1**, etc., up to **mydomaindata_0_999**.

11.4 How To Record Serialized Data

Scenario

Due to space limitations and speed, you want to store serialized data.

Configuration File

```
<dds>
<recorder name=”scenario4”>
...
    <domain name="mydomain">
        <domain_id> 98 </domain_id>
        <deserialize_mode>
           RTIDDS_DESERIALZEMODE_NEVER
        </deserialize_mode>
    </domain>
```

```
...
</recorder>
</dds>
```

Expected Outcome

All samples will be stored in a single column, along with SampleInfo and other metadata.

11.5 How To Record Using Best-Effort Reliability

Scenario

You have a system with multiple DataWriters of the same topic. Some of these use best-effort reliability, while others use strict reliability. You want to minimize the impact that the *Record* tool has on the system.

Configuration File

```
<dds>
<recorder name=”scenario5”>
...
    <topic_group name="Sensor">
        <topics>
            <topic_expr> Sensor* </topic_expr>
        </topics>
        <field_expr> *value* </field_expr>
        <datareader_qos>
           <reliability>
               <kind> BEST_EFFORT_RELIABILITY_QOS </kind>
           </reliability>
        </datareader_qos>
    </topic_group>
...
</recorder>
</dds>
```

Expected Outcome

The *Record* tool will use DataReaders with best-effort Reliability to record all data.

11.6 How To Enable Remote Access

Scenario

The *Record* tool is part of a larger system that must reach a steady state before it starts recording. The *Record* tool should use domain ID 54 and partition “rti” for communication with the controller.

Configuration File

```
<dds>
<recorder name=”scenario6”>
...
    <remote_access>
        <enabled> yes </enabled>
        <publish_status_period> 10 </publish_status_period>
        <remote_access_domain> domain54 </remote_access_domain>
        <subscriber_qos>
           <partition>
               <name>
                  <element> rti </element>
               </name>
           </partition>
        </subscriber_qos>
    </remote_access>
    <domain name="domain54">
        <domain_id> 54 </domain_id>
    </domain>

...
</recorder>
</dds>
```

Expected Outcome

The *Record* tool will communicate with a remote controller on domain ID 54 using partition “rti.” Status information will be published every 10 seconds.