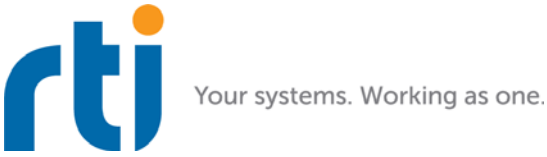
RTI Recording Service

User's Manual

Version 5.3.1



Printed in U.S.A. First printing.
February 2018.

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

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® DDS* application that records both *RTI Connext DDS* 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 10: Converting and Exporting Recorded Data.

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 can be configured to record from all partitions or 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 Connext DDS Core Libraries User's Manual.*

1.1 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

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

  The default installation paths are:

  - Mac® OS X® systems:

    /Applications/rti_connext_dds-*version*

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

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

  - UNIX-based systems, *root* user:

    /opt/rti_connext_dds-*version*

  - Windows® systems, user without Administrator privileges:

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

  - Windows systems, user with Administrator privileges:

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

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

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

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

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

  or if you have defined the NDDSHOME environment variable:

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

- **<path to examples>**

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

  Wherever you see <path to examples>, replace it with the appropriate path.

  By default, the examples are copied here:

  - Mac OS X systems:

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

- UNIX-based systems:

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

- Windows systems:

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

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

You can specify a different location for the **rti_workspace** directory. You can also specify that you do not want the examples copied to the workspace. See the *RTI Connext DDS Core Libraries Getting Started Guide*.

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 **<NDDSHOME>/bin**.

The tool should always be run using the executable script (**rtirecordingconsole.bat** for Windows systems, **rtirecordingconsole** for Linux systems). For Mac OS X systems, *Recording Console* includes a **.app** directory which includes a script to run the tool.

On Linux systems:

1. Open a command prompt and change to the **<NDDSHOME>/bin** directory
2. Start the *Console* by entering:
   > **rtirecordingconsole**
3. Set a Domain ID as described in Section 2.3.

On Mac OS X systems:

1. Go to the **<NDDSHOME>/bin** directory.
2. Execute the **RTI Recording Console.app** application by double-clicking on it or using Mac's **open** command.
3. If the *Connext DDS* bundle was installed under the Mac "Applications" directory, Mac's LaunchPad will also show the *RTI Recording Console* application in the **/bin** directory. Click on the application to run it.

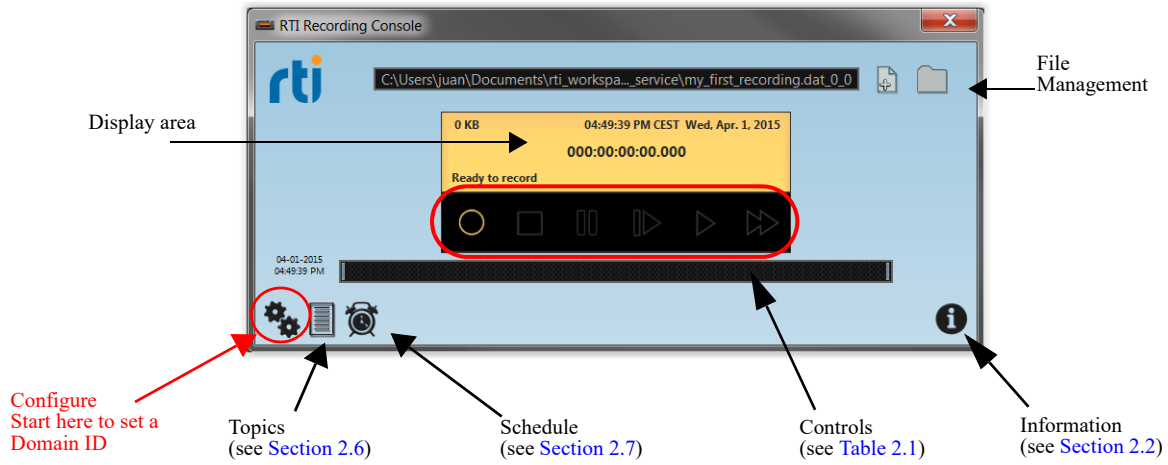
On Windows systems:

1. From the **Start** menu, navigate to **RTI Recording Service <version>**, and select **Recording Console**.
2. Set a Domain ID as described in Section 2.3.

Table 2.1 **Console's Controls**

					
Record	Stop	Pause/Resume	Single Step (Press Pause first)	Play	Fast Forward

Figure 2.1 **Console at Startup**

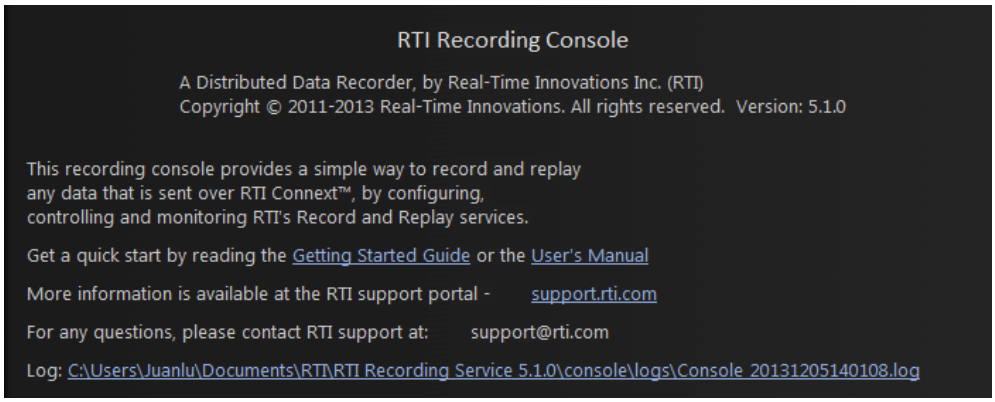


2.2 Using the Information Panel

To open the Information panel, click on the ⓘ in the *Console's* lower-right corner. From this panel, seen in Figure 2.2, you can access documentation, contact RTI Support, or view the most recent log file.

Note: Log files are automatically deleted when the *Console* shuts down, unless errors were reported.

Figure 2.2 **Information Panel**



The information panel provides links to documentation, RTI Support, and the Console's log file.

2.3 Configuring Recording Console

Before you can use *Recording Console* to record or replay data, you must specify a domain ID in the Configuration panels. The default domain ID is 0.

- To record data, specify the domain in which the data you want to record is being published.

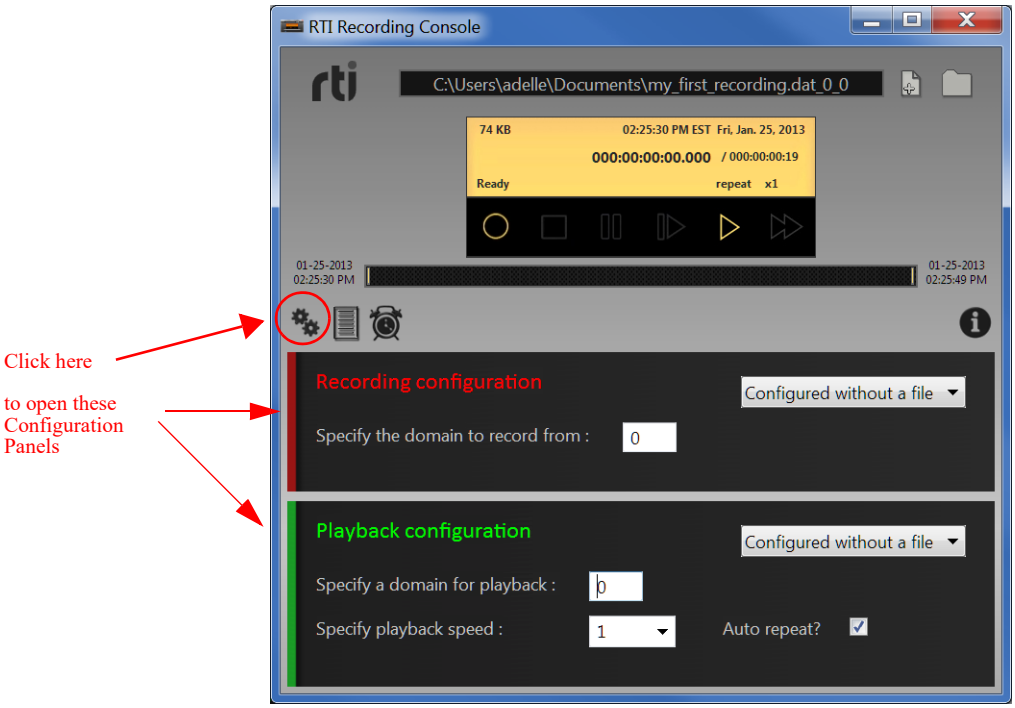
- To replay data, specify the domain that you want to publish the previously recorded data into.

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

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 used by *Recording Console* when recording or replaying the data.

There are two ways to configure *Recording Console*'s recording and playback parameters:

- Specify values in the Configuration panels shown below.
- Specify a file and a configuration within that file. This method is described in Section 2.3.1.



2.3.1 Configuring From an External File

If you have a use case that 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 that 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.

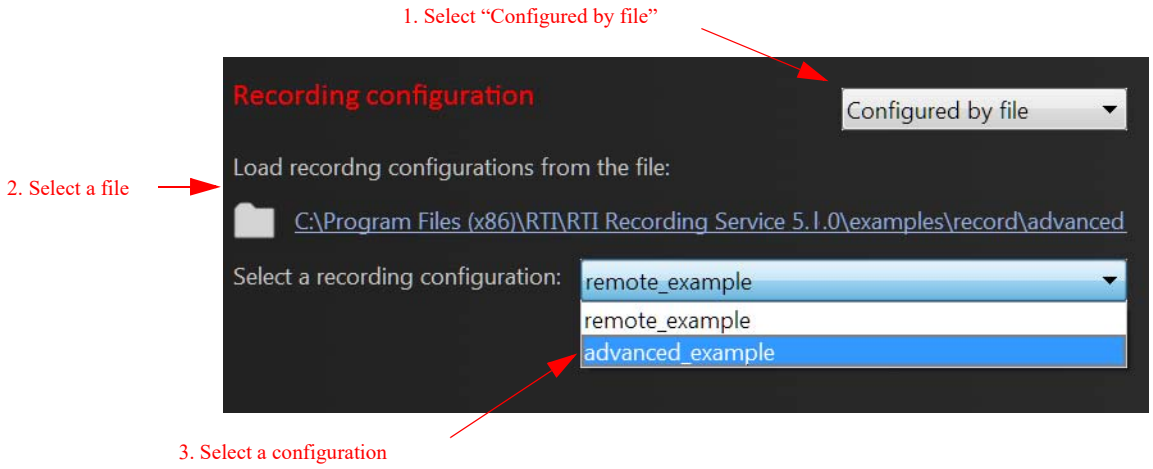
Note: Not all configuration settings are taken from the selected external file; see Section 2.3.1.1 and Section 2.3.1.3.

2.3.1.1 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, select **Configured by file**.

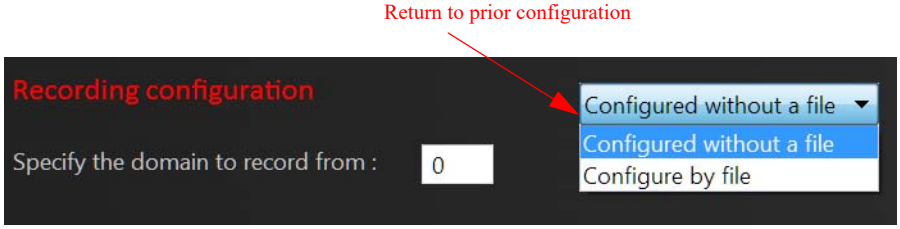
3. Select a configuration file to use for recording either by pressing the Open Folder button or by dragging-and-dropping a file from your file explorer window into the Recording configuration panel.
4. Select one of the configurations in the file by choosing from the drop-down listbox.



Only parts of the configuration file are used, while other settings are taken from *Recording Console*:

- <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 used, except as noted below:
  - <remote_access>: The domain ID is used, but 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 stop using the configuration file and go back to the previous QoS settings, select **Configured without a File**.



2.3.1.2 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:
  C:\Users\<*user name*>\Documents\rti_workspace\<*version*>\user_config\recording_service\settings.ini

- On Linux and OS X systems:
  ~/rti_workspace/<version>/user_config/recording_service/settings.ini

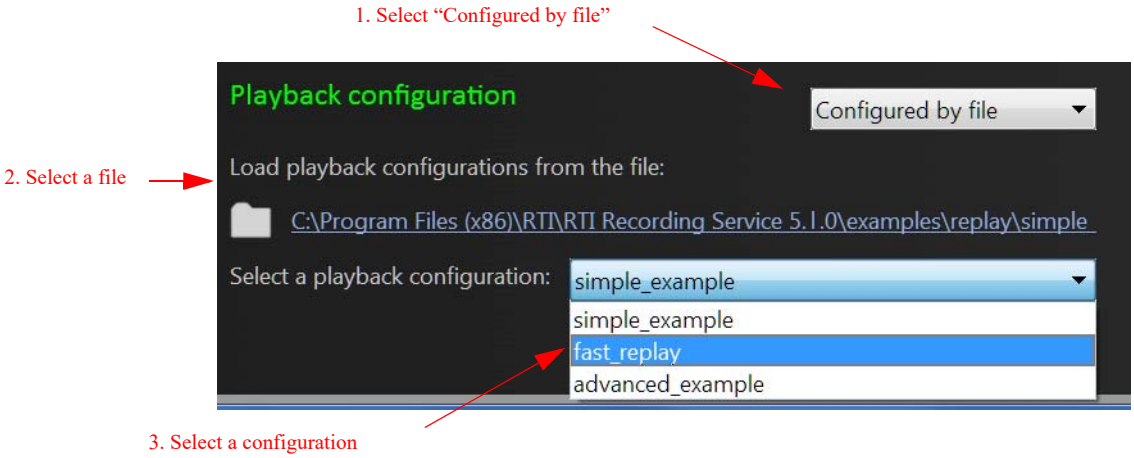
Again, this recommendation only applies if the domain in your configuration file is domain 99.

Note: Except for the specific changes described in this document, avoid making other changes in the **settings.ini** file.

2.3.1.3 Configuring Replay from an External File

To use an external configuration file for replay:

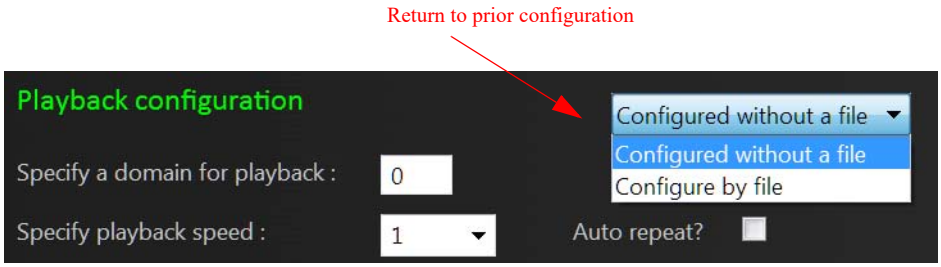
1. Press  to open the Recording and Playback Configuration panels.
2. In the Playback configuration panel, select **Configured by file**.
3. Press the Open Folder button  to select a configuration file for playback.
4. Select one of the configurations in the file by choosing from the drop-down listbox.
5. Select a QoS profile from the drop-down listbox.



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

- <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 used from this element, except for:
  - Only the first <session> from the first <replay_database> will be used.
  - The <filename> will be replaced with the file specified in *Recording Console*.

Note: To stop using the configuration file and go back to the previous QoS settings, select **Configure without a File**.

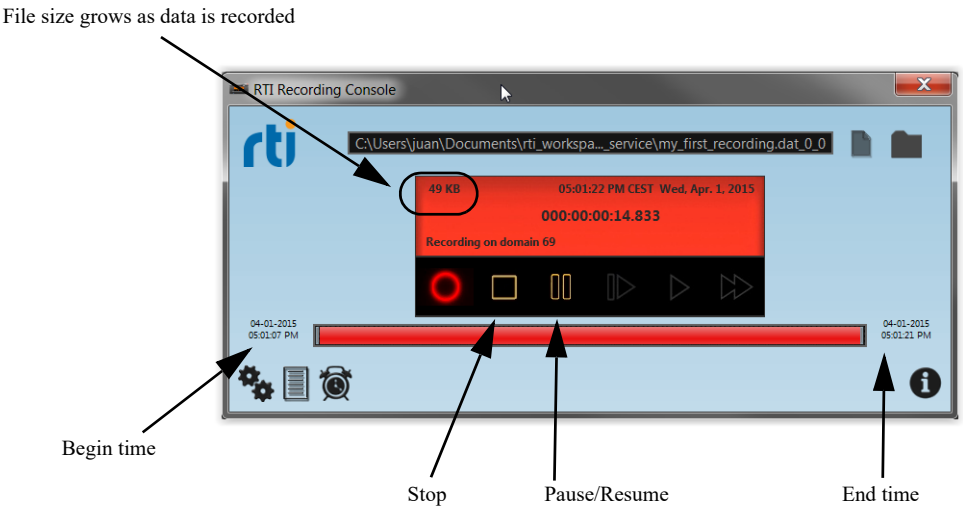


2.4 Recording Data

To record data:

1. Make sure you have set the domain ID in the Record Configuration panel (see Section 2.3).
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  in the upper-right corner 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 and locate the file that you want to record into.
     - Another way to open a recording file is simply to drag the file from your file explorer window and drop it into the long black rectangle at the top of the *Console*.

   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.
3. Press the Record button  to start recording.



2.4.1 Using the Pause/Resume Button During Recording

To pause recording, press the Pause/Resume button  .

To resume recording, either press Pause/Resume again or press Record  .

2.4.2 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 DDS* in its **<NDDSHOME>/bin** directory).
- You have access rights to create files in the directory where the recording file is to be created.

2.5 Replaying Data

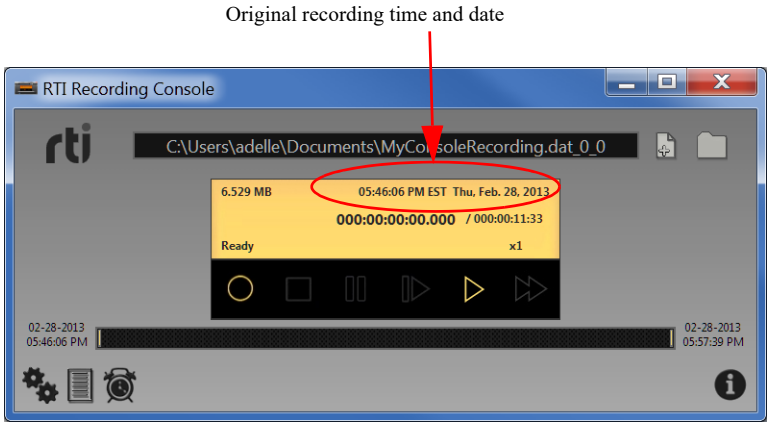
You can use the *Console* to replay data that was recorded using the *Console* or the *Record* tool. You may replay data recorded with an older version of *Recording Service*.

Note: If you recorded the data using the sqlite_ pragmas functionality described in Section 4.5, the resulting database cannot be replayed by *Recording Console;* use the *Replay* tool instead (see Chapter 6).

To replay data:

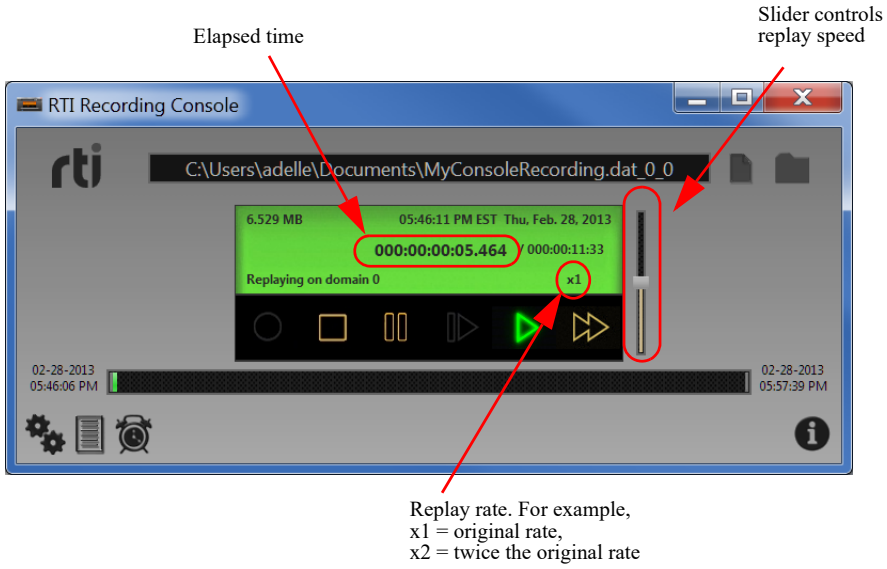
1. Make sure you have set the domain ID in the Playback Configuration panel (see Section 2.3).
2. Press the Open Folder button  in the upper-right corner, locate the file whose data is to be replayed, then click **Open**.
   - Another way to open a recorded data file is simply to drag the file from your file explorer window and drop it into the long black rectangle at the top of the *Console*.

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



3. Press Play  to begin replaying the data.

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



2.5.1 Using the Play Button

To begin replaying data, press Play .

2.5.2 Using the Fast-Forward Button

The Fast-Forward button will replay at a higher rate as long as the button is pressed. “Higher rate” means the next highest option from the list of playback speed choices in the drop-down menu. For example, if you were replaying at normal speed, fast-forward will replay at x2 (double) speed. If you were replaying at double speed, fast-forward will replay at x10 speed.

If replay is paused when you press Fast-Forward button, it will replay at the higher rate as long as the button is pressed. When you let go of the button, replay will go back to being paused.

If replay is not paused when you press Fast-Forward button, it will replay at the higher rate as long as the button is pressed. When you let go of the button, replay will return to the previous rate.

2.5.3 Using the Pause/Resume Button

To pause a replay, press the Pause/Resume button .

To resume the replay at the same rate, either press Pause/Resume again or press Play .

2.5.4 Advanced Configuration

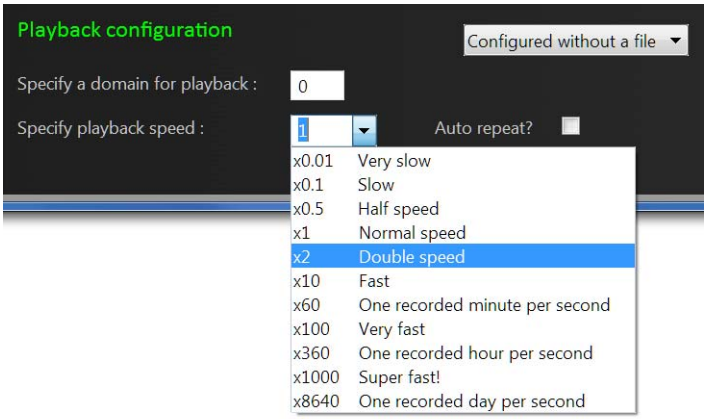
2.5.4.1 Changing the Replay Rate

To increase the rate *temporarily*, keep the Fast-Forward Button pressed (see Section 2.5.2).

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

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.

If you use the slider to change the playback speed, you can easily go back to default playback rate by pressing the Play button again while replay is in progress.

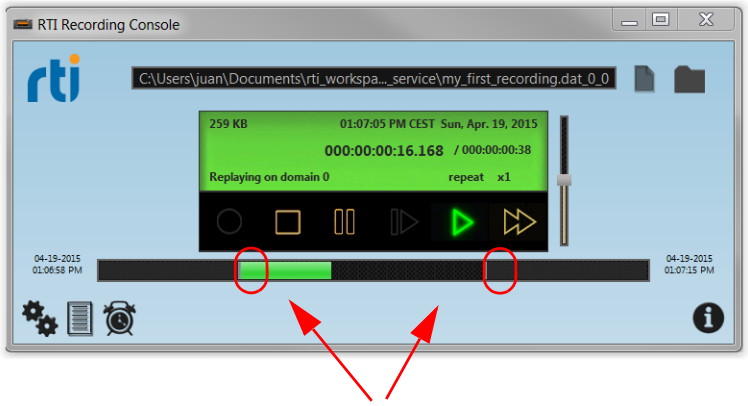


2.5.4.2 Auto-Repeat

To automatically repeat the replay in a continuous loop, select the **Auto repeat** check-box.

2.5.5 Restricting the Time Range to be Replayed

You can limit the start and end time for replaying data while replay is stopped by dragging the bars seen below:



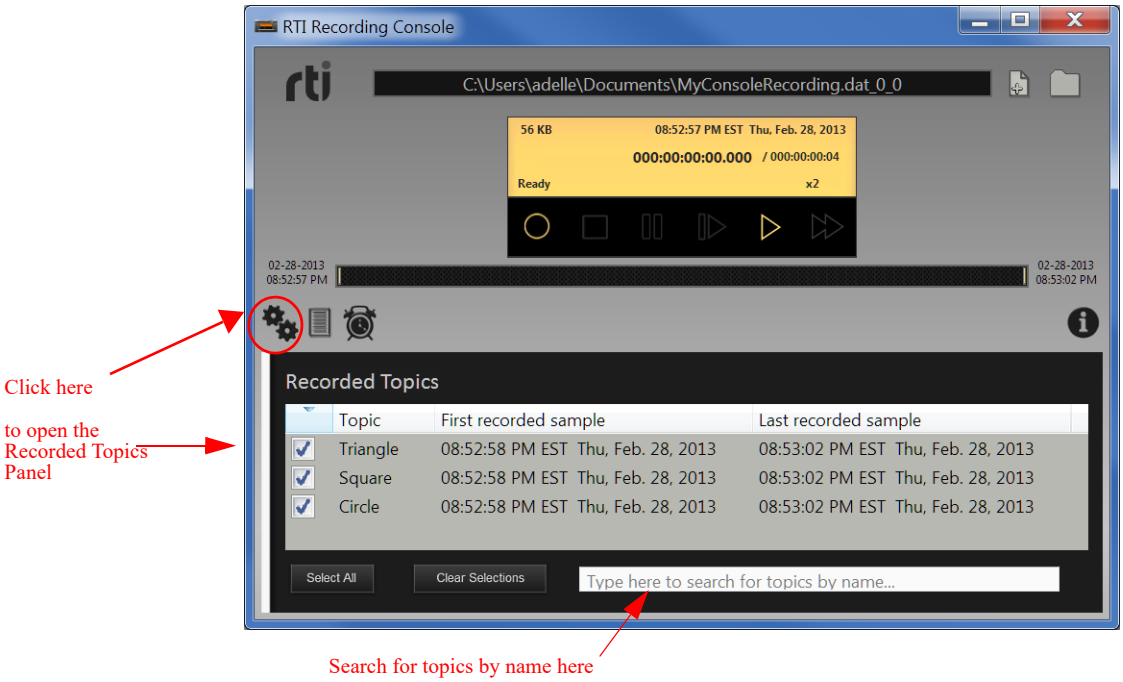
Drag these bars inward to restrict the time range for replaying data

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

2.6 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.

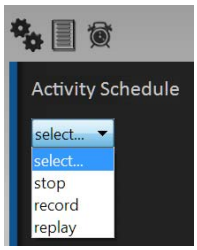
To restore and see the list of all topics, remove (erase) the search string from the search box.

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).

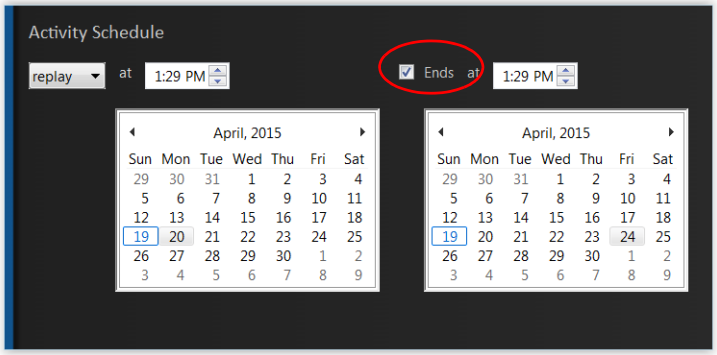
2.7 Scheduling Recording and Replay Tasks

To schedule recording or replay:

1. Press the Schedule button .
2. Select the type of task (record, replay, or stop current operation).
3. Select the starting time and date.



4. Optionally, select an ending time for the activity.



Important Notes:

- *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.

2.8 Troubleshooting

If *Recording Console* hangs, issues a generic error message, or behaves in an unexpected manner, here are some steps to resolve the problem:

1. Read the instructions again to ensure you understand the expected behavior.
2. It's possible that the environment or configuration in which *Recording Console* is running is causing the unexpected behavior. *Recording Console* generates a log file. To open the log file, click on the info button ('i') in the lower right corner, then click on the link at the bottom of the info panel. Each line in the log begins with a severity level. Look for lines that start with WARN or ERROR.
3. Contact RTI's support team.

   If you cannot resolve the issue based on the information in the log file, please contact us so we can help you overcome the issue.

   Before contacting support, please increase the logging verbosity and try to reproduce the problem so you can provide more detailed information. To do so:

   a. Close any running instances of *Recording Console* (otherwise, any change you make to the settings file may be overwritten).

   b. Open the console settings file in your home directory:

      - **rti_workspace/*version*/user_config/recording_service/settings.ini** (rti_workspace is described in Paths Mentioned in Documentation (Section 1.1))

      Note: Except for the specific changes described in this document, avoid making other changes in the **settings.ini** file.

   c. Find the line with "loggingLevel=INFO" and replace the default value of "INFO" with "TRACE".

   d. Save the updated settings file.

   e. Run *Recording Console* again until the issue you encountered before is reproduced.

   f. Find the log file. The info panel has a link indicating the name of the current log file.

In general, log files are stored here: **rti_workspace/*version*/user_config/recording_service/logs**.

Log files are named by date and time. Log files are only saved if there is an error. If no errors are reported, the log file is deleted when *Recording Console* shuts down.

g. Save a copy of the log file in a safe location and close *Recording Console*. Review the log file and remove any sensitive information that you would prefer not to expose.

h. Repeat the above steps to restore the logging level to the default "INFO" setting.

i. Contact RTI support. Click on the info button ('i') in the lower right corner of *Recording Console* to open the info panel, which contains links for contacting RTI support.

Provide a description of the problem, how to reproduce it, and a copy of the log file. If possible, please include a screenshot that demonstrates the problem.

RTI's support team is very responsive, you should receive a reply within 24 hours.

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 **<NDDSHOME>/bin** directory. Then enter:

- On Linux and Mac OS X systems:

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

- On Windows systems:

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

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

The *Record* tool is dynamically linked against the *Connext DDS* libraries. You should run the tool from the scripts in the **<NDDSHOME>/bin** directory—not from the executable files themselves. The scripts set all the paths and variables needed for the tool to find the shared libraries and run correctly.

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

```
$ rtirecord -listCfgs
```

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 **<path to examples>/recording_service** directory.

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

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 except when using the default configuration file. If using the default configuration file, do not use **-cfgFile**. 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.
-heapSnapshotPeriod	Enables heap monitoring. *Recording Service* will generate a heap snapshot every <sec>. Default: heap monitoring is disabled.
-heapSnapshotDir	When heap monitoring is enabled, this parameter configures the directory where the snapshots will be stored. The snapshot filename format is RTI_<configurationName><processId><index>.log. Default: current working directory
-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.
-listCfgs	Lists the available configuration profiles.
-verbosity	Specifies what type of logging information should be printed. 0: silent (*Connext DDS* and the *Record* tool) 1: errors (*Connext DDS* and the *Record* tool) (default) 2: warnings (the *Record* tool only) 3: warnings (*Connext DDS* and the *Record* tool) 4: information (the *Record* tool only) 5: tracing (the *Record* tool only) 6: tracing (*Connext DDS* 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.

3.3 Format of the Recorded Data

This section describes the format of the recorded data. For information on viewing the data, see Chapter 9: Viewing Recorded Data.

3.3.1 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 Connext DDS* C API Reference HTML documentation for the fields in each of the corresponding builtin topics. (In the HTML documentation for the C API, select **Modules, RTI Connext DDS API Reference, Domain Module, Built-in Topics**.)

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.

The fields stored for each discovery table are listed in Appendix A. If no field selection settings enabled, the *Record* tool will just create the DCPSPublication table with enough information for the *Replay* tool to work. Fields can be added to the tables, see Choosing Which SampleInfo and Discovery Fields to Record (Section 4.5.1). If fields are added to the DCPSParticipant and DCPSSubscription tables they will be created.

3.3.2 User Data

3.3.2.1 Deserialized Data

When the *Record* tool stores data in deserialized form, it creates a mapping from a Topic's type 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;
};
```

3.3.2.2 Serialized Data

If the topic data is saved in serialized form, the user data table will contain the following columns:

- **rti_serialized_sample**: Raw binary data for the sample
- **rti_serialized_length**: Length of the raw data blob
- **rti_serialized_endian**: 1 — little endian, 0 — big endian

3.3.2.3 Sample Info and Metadata Fields

The *Record* tool creates a table called **TopicName$RecordGroupName$Domain-Name** 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 extra columns, by default:

- The Sample Info's reception timestamp (in nanoseconds since Jan. 1st, 1970).
- The Sample Info's valid data boolean indicator.
- Table_prefix: A string representing "RecordGroupName$DomainName" (**only** if <shared_table> is specified—see TopicGroup Properties (Section 4.8)).

The fields selected for recording can be modified using the <user_topic_metadata_fields> tag in both the Database (Output File) Properties (Section 4.5)) and RecordGroup Properties (Section 4.9) in the Recorder configuration. The name and SQL type for all the fields that can be recorded appear in Appendix A.

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 DDS* (see the chapter on Configuring QoS with XML in the *RTI Connext DDS Core Libraries User's Manual*).

- **<NDDSHOME>/resource/xml/NDDS_QOS_PROFILES.xml**

  This file contains the *Connext DDS* 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.

The next locations are specific to *Recording Service*.

- **<NDDSHOME>/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 XSD definitions followed by the configuration are in the document **resource/schema/rti_record.xsd**. All the following main sections are optional; if specified, they must be in this order:

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

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. -->
        <recorder_database>
            <database_name> simple_config.dat </database_name>
        </recorder_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 DDS*. Please see the *RTI Connext DDS Core Libraries 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.

Table 4.1 **Property Value Data Types**

Type	Format	Notes
DDS_Enum	A string	Enum values are not case sensitive. Legal values are those listed for the property in the *RTI Connext DDS* C API Reference HTML documentation.
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.

DDS_QosPolicy	See the *RTI Connext DDS* C API Reference HTML documentation for the structure of each QoS policy, and the *RTI Connext DDS Core Libraries 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 *Connext DDS* C API.
		For enumerations, the legal constants are those defined for the *Connext DDS* 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.)

Table 4.1 **Property Value Data Types**

Type	Format	Notes
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>	*Deprecated. See auto_start_mode below.* 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
auto_start_mode	<auto_start_mode> RECORD_ENABLED | DISCOVERY_RECORD_ENABLED | RECORD_DISABLED </auto_start_mode>	Whether or not the *Record* tool should start recording data right away (user-data or discovery data) after initialization. This option is useful only when the *Record* tool is administered remotely. Valid options are: • RECORD_ENABLED: The *Record* tool will start recording all data (user-data and discovery). This is equivalent to old auto_start=true. • DISCOVERY_RECORD_ENABLED: The *Record* tool will start recording discovery data but will not record user-data samples. • RECORD_DISABLED: No recording to the DB will happen. This is equivalent to old auto_start = false. If you are going to use *Replay* or *Converter* with the recorded database files, the advice is to use DISCOVERY_RECORD_ENABLED at a minimum. Discovery can occur while *Recording Service* is not recording data to the database. If this discovery data is not recorded, then *Replay* or *Converter* may find that some necessary information in the DCPSPublication table is missing and thus may not function correctly. Use RECORD_DISABLED only if really nothing should be stored in the database until the START command is sent. Default: RECORD_ENABLED See also: RECORDER_ CONFIGURE in Table 5.1, "Messages Types Exchanged Between Record Tool and Remote Access Application," on page 5-6.

Table 4.2 **General Properties**

Property	Syntax	Description
replay_ compatibility	<replay_compatibility> *DDS_Boolean* </replay_compatibility>	If set to true, the *Record* tool will always preserve the fields needed by the *Replay* tool to work with the recorded database, regardless of the field selection settings (see Table 4.4, "Database Properties," on page 4-10 and Table 4.15, "RecordGroup Properties," on page 4-35) specified by the user. 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 Remote Access Properties

As you will see in Chapter 5: Accessing the Record Tool from a Remote Location, you can create a *Connext DDS* 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 4.3, "Remote Access Properties," on page 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.

4.4.1 Enabling RTI Distributed Logger in the Record Tool

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

Distributed Logger is included in *Connext DDS* but it is not supported on all platforms; see the *RTI Connext DDS Core Libraries Platform Notes* to see which platforms support *Distributed Logger*.

When you enable *Distributed Logger,* the *Record* tool will publish its log messages to *Connext DDS*. Then you can use *RTI Monitor*[1] to visualize the log message data. Since the data is provided in a *Connext DDS* 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.

Table 4.3 **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.
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 4.4.1).
participant_qos	<participant_qos> *DDS_DomainParticipantQos* </participant_qos>	When the *Record* tool is configured to use a dedicated participant for administration (see remote_access_domain in this table), this tag configures the QoS for that dedicated DomainParticipant. This tag is ignored if the remote_access_domain tag is configured to use one of the *Record* tool's user-defined domains (<domain> tag). Default: Default Domain Participant QoS settings.
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.

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

Table 4.3 **Remote Access Properties**

Properties under <remote_access>	Syntax	Description
remote_access_domain	<remote_access_domain> *String* </remote_access_domain>	***Required.*** Specifies which domain the *Record* tool will use to enable remote access. Only one domain can be specified. There are two ways to use this field: • If a domain ID (positive number) is specified, the *Record* tool will create a dedicated Domain Participant, which will be used for the tool's remote administration. This DomainParticipant will be created apart from the pool of user-defined domains. • If a domain reference (the name of a user-defined domain, e.g., <domain name="use this name">...</domain>) is specified, the *Record* tool will use the user-defined DomainParticipant referenced by name. **Note:** using this format has implications. When you stop the Record tool, domain administration is preserved. This means that its DomainParticipant is not deleted; thus when the tool is restarted, rediscovery (of the entities created before stopping) won't happen for that particular domain. This can cause discovery data to be missing for that domain in the discovery tables. Hence, this option is not recommended if the domain is also going to be used to record user data. Default: Undefined
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.

```
<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 *Distributed Logger* section of the *RTI Connext DDS Core Libraries User's Manual.*

4.5 Database (Output File) Properties

Table 4.4, "Database Properties," on page 4-10 describes the Database properties. All database properties are optional except **database_name**. All database properties must be specified within **<recorder_data-**

base> and
</recorder_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®.

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.

Table 4.4 **Database Properties**

Properties under <recorder_database>	Syntax	Description
builtin_topic_metadata_fields	<builtin_topic_metadata_fields> <DCPSParticipant_topic> *ParticipantData fields* </DCPSParticipant_topic> <DCPSPublication_topic> *PublicationData fields* </DCPSPublication_topic> <DCPSSubscription_topic> *SubscriptionData fields* </DCPSSubscription_topic> </builtin_topic_metadata_fields>	Specifies which fields to include/ exclude in the discovery tables. There are settings for each of the three recorded discovery tables described in Table 4.5.
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>

Table 4.4 **Database Properties**

Properties under <recorder_database>	Syntax	Description
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>
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 4.4 **Database Properties**

Properties under <recorder_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 follow the "TopicName$RecordGroupName$DomainName" convention and fields in Topics use **$** to navigate hierarchical types, such as **abc**.. The dollar sign (‘**$**’) is used as the default path separator instead of the more conventional comma (',') because **$** does not require quotes when used in SQLite SQL statements. For example, to use '**#**' as the path separator: <path_separator> # </path_separator> **Notes:** • Using a character that can be included in a field name, such as an "_" (underscore) may lead to errors when you try to replay the file with *Replay* if any field contains that character. • This is not a required property, but when added, it 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 max_file_size limit has been reached for the last file segment. • True: Overwrite existing file segments as needed (starting with the first one). • False: Stop recording data. Default: False
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.

Table 4.4 **Database Properties**

Properties under <recorder_database>	Syntax	Description
sqlite_pragmas	<sqlite_pragmas> <pragma> pragma name [plus assignment or parameters] </pragma> ... </sqlite_pragmas>	Specifies a list of SQLite pragma statements to be applied to the database. There must be at least one '<pragma>' object in the list; there is no upper limit for the number of pragmas that can be specified. Pragmas are applied right after database creation and before any table is created. The pragma name is just the name of the SQLite pragma (e.g., **page_size**) as defined by SQLite. The **PRAGMA** SQLite keyword should not be specified, as the *Record* tool will do that automatically. Values or parameters can be provided (e.g., **page_size = 4096** or **table_info(DCPSPublication**)). The output of the pragma statements is shown in standard output. See https://www.sqlite.org/pragma.html Notes: • One possible pragma is the database journal mode. One of the journal modes is WAL (write-ahead logging). This mode improves concurrency when reading and writing to the same database file from different processes or threads. However, WAL doesn't work on network file systems. • The order in which you specify the pragmas is the order in which they will be applied.
user_topic_metadata_fields	<user_topic_metadata_fields> <included> <field> field expression </field> ... </included> <excluded> <field> field expression </field> ... </excluded> </user_topic_metadata_fields>	Specifies lists of included and excluded Sample Info or metadata fields in the user topic tables. A field expression is a Posix fnmatch expression to match field names as described in Appendix A (e.g. "**SampleInfo_***" will match all Sample Info specific fields). There is no upper limit to the number of **<field>** expressions to be specified. The 'included' and 'excluded' fields are optional. If both are defined, the 'included' expressions are processed before the 'excluded' ones. User topic field settings can also be expressed in the Record Group settings (see Table 4.15, "RecordGroup Properties," on page 4-35). If defined, Record Group settings take precedence over general database settings defined here.

4.5.1 Choosing Which SampleInfo and Discovery Fields to Record

To reduce database size and increase flexibility, the *Record* tool provides a way to select which Sample Info, discovery (DCPSPublication, DCPSSubscription and DCPSParticipant), and metadata fields should

Table 4.5 **Builtin Topic (Discovery) Field Selection Properties**

Properties under <builtin_topic_metadata_fields>	Syntax	Description
DCPSParticipant_topic	<DCPSParticipant_topic> <included> <field> field expression </field> ... </included> <excluded> <field> field expression </field> ... </excluded> </DCPSParticipant_topic>	Specifies lists of included and excluded fields in the DCPSParticipant table. A field expression is a POSIX fnmatch expression to match field names as described in Appendix A (e.g. "ParticipantData_*" will match all DCPSParticipant specific fields). There is no upper limit to the number of <field> expressions to be specified. The 'included' and 'excluded' fields are optional. If both are defined, the 'included' expressions are processed before the 'excluded' ones.
DCPSPublication_topic	<DCPSPublication_topic> <included> <field> field expression </field> ... </included> <excluded> <field> field expression </field> ... </excluded> </DCPSPublication_topic>	Specifies lists of included and excluded fields in the DCPSPublication table. A field expression is a POSIX fnmatch expression to match field names as described in Appendix A (e.g. "PublicationData_*" will match all DCPSPublication specific fields). There is no upper limit to the number of <field> expressions to be specified. The 'included' and 'excluded' fields are optional. If both are defined, the 'included' expressions are processed before the 'excluded' ones.
DCPSSubscription_topic	<DCPSSubscription_topic> <included> <field> field expression </field> ... </included> <excluded> <field> field expression </field> ... </excluded> </DCPSSubscription_topic>	Specifies lists of included and excluded fields in the DCPSSubscription table. A field expression is a POSIX fnmatch expression to match field names as described in Appendix A (e.g. "SubscriptionData_*" will match all DCPSSubscription specific fields). There is no upper limit to the number of <field> expressions to be specified. The 'included' and 'excluded' fields are optional. If both are defined, the 'included' expressions are processed before the 'excluded' ones.

be recorded. An application may not be interested in recording all the metadata, SampleInfo, and discovery information.

Before this feature was introduced, the *Record* tool would record everything by default. Now, the *Record* tool will only record the fields necessary for the *Replay* tool to work with the database. These fields are:

- In the user table:
  SampleInfo_valid_data and **SampleInfo_reception_timestamp**.
- In the DCPSPublication table:
  PublicationData_topic_name, **PublicationData_type_name**, **PublicationData_typecode** and **PublicationData_typecode_length**.

The *Record* tool provides capabilities to add to, or to remove from, this set of fields. It also provides a special boolean flag called **replay_compatibility** (see Table 4.2, "General Properties," on page 4-6). When this flag is enabled (the default), it preserves the above set of fields, regardless of which fields the user chooses to exclude. This way very general regular expressions can be used without affecting compatibility with the *Replay* tool.

Table 4.4 describes the XML settings for selecting fields in the discovery tables (tag <builtin_topic_metadata_fields>). There are settings for each of the three discovery tables created by the *Record* tool (tags <DCPSParticipant_topic>, <DCPSPublication_topic> and <DCPSSubscription_topic>). In any of the tables, if you decide to exclude all fields (for example: <excluded><field> * </field></excluded>), the table will not be created; that is, if a table would end up with no columns, the *Record* tool won't create it.

The field selection settings also apply to user topic tables. The tag used to define those settings is called <user_topic_metadata_fields>; it can be used inside <recorder_database> to define general settings, and inside <record_group> to define settings that apply only to a specific Record Group. Record Group settings take precedence over general database settings. These settings are described in Table 4.4, "Database Properties"and Table 4.15, "RecordGroup Properties," on page 4-35.

4.5.1.1 Field Selection Examples

This section shows some examples of how to configure field selection settings for the *Record* tool. The following is a basic recording configuration used as the starting point for the examples. It is based on the default configuration shipped with the product (see the file **RTI_RECORDING_SERVICE.xml** in the **resource/xml** directory):

```
<recorder name="basic">
    <!-- The replay compatibility flag tells the Record tool that
         the fields needed by Replay must be preserved
         regardless of the field selection configuration -->
    <replay_compatibility> true </replay_compatibility>

        <remote_access>
            <enabled> true </enabled>
            <remote_access_domain> domain0 </remote_access_domain>
            <distributed_logger>
                <enabled> true </enabled>
                <filter_level> WARNING </filter_level>
            </distributed_logger>
        </remote_access>

        <recorder_database>
            <database_name> rti_recorder_default.dat
            </database_name>
        </recorder_database>

        <domain name="domain0">
            <domain_id> 0 </domain_id>
            <deserialize_mode> RTIDDS_DESERIALIZEMODE_ALWAYS
            </deserialize_mode>
        </domain>
```

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

            <record_group name="RecordAll">
                <domain_ref>
                    <element> domain0 </element>
                </domain_ref>
                <topic_ref>
                    <element> AllTopics </element>
                </topic_ref>
            </record_group>
        </recorder>
```

By default, the *Record* tool will record only the fields that the *Replay* tool will need to work with the recorded database. Furthermore, because the **replay_compatibility** flag is set to true, these fields will be protected by the *Record* tool and will always be present when recording.

4.5.1.2 Example 1: Record Everything

The following configuration uses the **builtin_topic_metadata_fields** and **user_topic_metadata_field** settings in the **recorder_database** section to record *all* possible fields in *all* tables, including the discovery ones.

```
<recorder name="basicAllExtraFields">
<!-- The replay_compatibility flag tells the Record tool that
     the fields needed by the Replay tool must be preserved
     regardless of the field selection configuration. In this
     case, it's not really necessary to set it to true, because
     we're going to ADD fields to the configuration, not REMOVE
     them -->
    <replay_compatibility> true </replay_compatibility>

    <remote_access>
        <enabled> true </enabled>
        <remote_access_domain> domain0 </remote_access_domain>
        <distributed_logger>
            <enabled> true </enabled>
            <filter_level> WARNING </filter_level>
        </distributed_logger>
    </remote_access>

    <recorder_database>
        <database_name> rti_recorder_default.dat
        </database_name>

        <!-- Field selection settings for the discovery tables
             (DCPSParticipant, DCPSPublication, and
             DCPSSubscription). We use the '*' regular expression
             in the 'included' sections to indicate that all fields
             are to be recorded -->
        <builtin_topic_metadata_fields>
            <DCPSParticipant_topic>
                <included>
                    <field> * </field>
```

```
                    </included>
                </DCPSParticipant_topic>
                <DCPSPublication_topic>
                    <included>
                        <field> * </field>
                    </included>
                </DCPSPublication_topic>
                <DCPSSubscription_topic>
                    <included>
                        <field> * </field>
                    </included>
                </DCPSSubscription_topic>
            </builtin_topic_metadata_fields>

            <!-- Field selection settings for user topic tables.
                 The '*' field expression in the 'included'
                 settings tells the Record tool to record all fields
                 (Sample Info and metadata) -->
            <user_topic_metadata_fields>
                <included>
                    <field> * </field>
                </included>
            </user_topic_metadata_fields>
        </recorder_database>

        <domain name="domain0">
            <domain_id> 0 </domain_id>
            <deserialize_mode> RTIDDS_DESERIALIZEMODE_ALWAYS
            </deserialize_mode>
        </domain>

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

        <record_group name="RecordAll">
            <domain_ref>
                <element> domain0 </element>
            </domain_ref>
            <topic_ref>
                <element> AllTopics </element>
            </topic_ref>
        </record_group>
    </recorder>
```

4.5.1.3 Example 2: Record No Extra Fields

The following configuration will record only user data fields in user data tables. No Sample Info or metadata fields will be recorded. Moreover, no discovery information is selected to be recorded so no discovery tables will be created. The **replay_compatibility** flag must be disabled for this configuration to work, because if enabled, it would lock the *Replay* tool's compatibility fields and they would be recorded.

```
    <recorder name="basicNoExtraFields">
        <!-- The replay_compatibility flag must be false in this
             configuration, otherwise it would block removal of the
             Replay tool compatibility set of fields -->
```

```
<replay_compatibility> false </replay_compatibility>

<remote_access>
    <enabled> true </enabled>
    <remote_access_domain> domain0 </remote_access_domain>
    <distributed_logger>
        <enabled> true </enabled>
        <filter_level> WARNING </filter_level>
    </distributed_logger>
</remote_access>

<recorder_database>
    <database_name> rti_recorder_default.dat
    </database_name>

    <!-- Field selection settings for the discovery tables
         (DCPSParticipant, DCPSPublication, and
         DCPSSubscription). The '*' regular expression in
         the 'excluded' sections tells the Record tool
         that ALL fields are to be skipped -->
    <builtin_topic_metadata_fields>
        <DCPSParticipant_topic>
            <excluded>
                <field> * </field>
            </excluded>
        </DCPSParticipant_topic>
        <DCPSPublication_topic>
            <excluded>
                <field> * </field>
            </excluded>
        </DCPSPublication_topic>
        <DCPSSubscription_topic>
            <excluded>
                <field> * </field>
            </excluded>
        </DCPSSubscription_topic>
    </builtin_topic_metadata_fields>

    <!-- Field selection settings for user topic tables.
         Using the '*' field expression in the 'excluded'
         settings tells the Record tool that none of the
         extra fields (Sample Info and metadata)
         should be recorded. User-data fields will always
         be recorded -->
    <user_topic_metadata_fields>
        <excluded>
            <field> * </field>
        </excluded>
    </user_topic_metadata_fields>
</recorder_database>

<domain name="domain0">
    <domain_id> 0 </domain_id>
    <deserialize_mode> RTIDDS_DESERIALIZEMODE_ALWAYS
    </deserialize_mode>
</domain>

<topic_group name="AllTopics">
   <topics>
```

```
                <topic_expr> * </topic_expr>
            </topics>
            <field_expr> * </field_expr>
        </topic_group>

        <record_group name="RecordAll">
            <domain_ref>
                <element> domain0 </element>
            </domain_ref>
            <topic_ref>
                <element> AllTopics </element>
            </topic_ref>
        </record_group>
    </recorder>
```

4.5.1.4 Example 3: Overriding Database Settings with Record Group Settings

In the following configuration, two sets of settings are defined for user topic tables. The settings in the general database section include the **SampleInfo_source_timestamp** field in the user topic field selection settings (apart from the fields included by default). There is an extra Topic Group called **KeyedTopics**, which the user will associate with keyed topics. There may be an interest in recording the instance handle field (**SampleInfo_instance_handle**) for these topics. This may be done by defining a new Record Group with different field selection settings, as shown below.

One important aspect of the configuration is the use of the **exemption** expression in the **AllTopicsExceptKeyed** Topic Group. By using the exemption expression '**Keyed***' we make the two available Topic Groups (**AllTopicsExceptKeyed** and '**KeyedTopics**) disjoint, because **AllTopicsExceptKeyed** will never match the same topics as **KeyedTopics**. This is very important to be sure that the right settings apply to the right topics.

```
<recorder name="recordGroupOverriding">
    <!-- The replay_compatibility flag is set to true for this
         example although we only include fields, so it has no
         real effect -->
    <replay_compatibility> true </replay_compatibility>

    <remote_access>
        <enabled> true </enabled>
        <remote_access_domain> domain0 </remote_access_domain>
        <distributed_logger>
            <enabled> true </enabled>
            <filter_level> WARNING </filter_level>
        </distributed_logger>
    </remote_access>

    <recorder_database>
        <database_name> rti_recorder_default.dat </database_name>
        <!-- Field selection settings for user topic tables.
             We use the name of the source timestamp field in
             the 'included' section so it will be recorded
             with every user data sample -->
        <user_topic_metadata_fields>
            <included>
                <field> SampleInfo_source_timestamp </field>
            </included>
        </user_topic_metadata_fields>
    </recorder_database>

    <domain name="domain0">
```

```
            <domain_id> 0 </domain_id>
            <deserialize_mode> RTIDDS_DESERIALIZEMODE_ALWAYS
            </deserialize_mode>
        </domain>

        <topic_group name="AllTopicsExceptKeyed">
            <topics>
                <topic_expr> * </topic_expr>
                <!-- IMPORTANT: By using an exemption expression, we
                      make the 2 available Topic Groups disjoint -->
                <exemption> Keyed* </exemption>
            </topics>
            <field_expr> * </field_expr>
        </topic_group>

        <topic_group name="KeyedTopics">
            <topics>
                <!-- IMPORTANT: The match expression for this Topic
                     Group is exactly the exemption expression in
                     the other Topic Group ('AllTopicsExceptKeyed') -->
                <topic_expr> Keyed* </topic_expr>
            </topics>
            <field_expr> * </field_expr>
        </topic_group>

        <!-- The following Record Group will inherit the user topic
             field selection settings in the database configuration,
             meaning that it will record only the Replay compatibility
             fields plus the SampleInfo_source_timestamp fields -->
        <record_group name="RecordAllExceptKeyed">
            <domain_ref>
                <element> domain0 </element>
            </domain_ref>
            <topic_ref>
                <element> AllTopicsExceptKeyed </element>
            </topic_ref>
        </record_group>

        <record_group name="RecordKeyed">
            <domain_ref>
                <element> domain0 </element>
            </domain_ref>
            <topic_ref>
                <element> KeyedTopics </element>
            </topic_ref>

            <!-- Specific field selection settings for this Record
                 Group. These settings will override the general
                 settings in the database section.
                 So 'SampleInfo_instance_handle' will be recorded
                 but 'SampleInfo_source_timestamp' won't -->
            <user_topic_metadata_fields>
                <included>
                    <field> SampleInfo_instance_handle </field>
                </included>
            </user_topic_metadata_fields>
        </record_group>
    </recorder>
```

4.5.1.5 Example 4: Recording Everything Except the Metadata Fields

The following example configuration shows how to record all fields except the metadata fields (prefixed with **Metadata_**). The key for this is to first include everything with the '*' regular expression and then exclude fields matching **Metadata_***.

```
<recorder name="basicAllButMetadata">
    <!-- The replay compatibility flag tells the Record tool that
         the fields needed by the Replay tool must be preserved
         regardless of the field selection configuration. In this
         case, it's not really necessary to set it to true,
         because we're going to ADD fields to the configuration,
         not REMOVE them -->
    <replay_compatibility> true </replay_compatibility>

    <remote_access>
        <enabled> true </enabled>
        <remote_access_domain> domain0 </remote_access_domain>
        <distributed_logger>
            <enabled> true </enabled>
            <filter_level> WARNING </filter_level>
        </distributed_logger>
    </remote_access>

    <recorder_database>
        <database_name> rti_recorder_default.dat </database_name>

        <!-- Field selection settings for the discovery tables
             (DCPSParticipant, DCPSPublication, and
             DCPSSubscription). The '*' regular expression in the
             'included' sections indicates that all fields are to
             be recorded. By adding the expression 'Metadata_*' to
             the excluded fields list, we are removing the
             metadata fields -->
        <builtin_topic_metadata_fields>
            <DCPSParticipant_topic>
                <included>
                    <field> * </field>
                </included>
                <excluded>
                    <field> Metadata_* </field>
                </excluded>
            </DCPSParticipant_topic>

            <DCPSPublication_topic>
                <included>
                    <field> * </field>
                </included>
                <excluded>
                    <field> Metadata_* </field>
                </excluded>
            </DCPSPublication_topic>

            <DCPSSubscription_topic>
                <included>
                    <field> * </field>
                </included>
                <excluded>
                    <field> Metadata_* </field>
```

```
                </excluded>
            </DCPSSubscription_topic>
        </builtin_topic_metadata_fields>

        <!-- Field selection settings for user topic tables.
             The '*' field expression in the 'included' settings
             tells the Record tool to record all fields (Sample Info
             and metadata). Then by adding the expression
             'Metadata_*' to the excluded fields list, we remove
             the metadata fields -->
        <user_topic_metadata_fields>
            <included>
                <field> * </field>
            </included>
            <excluded>
                <field> Metadata_* </field>
            </excluded>
        </user_topic_metadata_fields>
    </recorder_database>

    <domain name="domain0">
        <domain_id> 0 </domain_id>
        <deserialize_mode> RTIDDS_DESERIALIZEMODE_ALWAYS
        </deserialize_mode>
    </domain>

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

    <record_group name="RecordAll">
        <domain_ref>
            <element> domain0 </element>
        </domain_ref>
        <topic_ref>
            <element> AllTopics </element>
        </topic_ref>
    </record_group>
</recorder>
```

4.6 Domain Type Configuration

The top-level tag <domain_type_config> allows you to pass type configuration information to the *Record* and *Converter* tools in the form of XML type-configuration files. Table 4.6 describes the Domain Type Config properties. All Domain Type Config properties are optional.

Table 4.6 **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 Domain Properties (Section 4.7)). These type configuration elements can be repeated. For more details in the contents of this tag See Table 4.7, "Domain Group Configuration Properties," on page 4-23.

Table 4.7 **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 Domain Properties (Section 4.7).
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 4.8, "Type Config Properties".

Table 4.8 **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 4.9, "XML Type Configuration Properties".

Table 4.9 **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 4.10, "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 4.10 **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.
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 4.9, "XML Type Configuration Properties"). Default: TRUE.
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 the *Record* tool in the DomainParticipants. See Table 4.11, "Type Registration Properties"

Table 4.11 **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.
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.
topics	<topics> <element> *Topic name string* </element> <topics>	This is a list of regular POSIX fn-name expressions (every <element> entry is equivalent to an expression). A topic whose name matches any of the expressions will be recorded using the type definition designated by <type_name>. If a topic matches more than one expression in different <type> entries, the first one that was matched will be used.
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.

4.6.1 Automatic Mode and the Use of XML Types

The *Record* tool offers an automatic recording mode, RTIDDS_DESERIALIZEMODE_AUTOMATIC (see the <deserialize_mode> tag in Table 4.12, “Domain Properties,” on page 4-26). When using this mode, if no type-code is found for a type, the *Record* tool will record the type in serialized mode and interpret the sample as if it was a raw bytes sample. While this mode can be useful, it can also be dangerous because there is no indication—not in the DCPSPublication table or in XML configuration—of the original type. The *Record* tool will record the type but won't record any information about the original type because there wasn't any. When the time comes to extract the data, both *Converter* and the *Replay* tool need the original type. This means the type has to be provided anyway, and because it's not in the database, it can only be provided via XML types.

There are some disadvantages to this approach. First, types evolve as systems evolve. A database can be recorded at some point but the need to extract, use, and examine the data may not happen until later on. If the type provided to *Converter* or the *Replay* tool in the absence of recorded discovery types does not match the type used to record the data, there could potentially be errors while trying to extract the data. There is nothing we can do about that except to provide the right type version, which can become quite cumbersome if there is no version control in place, for example.

Second, the encoding of CDR samples is different for mutable types and types with optional members than it is for extensible and final types. But the raw bytes type used to record plain serialized data doesn't take this into account. So there can potentially be problems with this approach. New versions of the *Record* tool flag this and new versions of the *Replay* tool and *Converter* can work with that flag, but this doesn't apply to databases recorded with older versions.

As a rule of thumb, we generally recommend that for types for which no type information is shared via discovery, you provide XML types in the Recorder configuration. While XML types are necessary to record in deserialized format, and in serialized format when the mode is set to DESERIALIZE_NEVER, they are not necessary to record in serialized format when using the automatic mode mentioned above.

4.7 Domain Properties

Table 4.12 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 4.9), 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).

Table 4.12 **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 4.7.2).
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 Connext DDS Core Libraries User's Manual* for details. (See the chapter on Configuring QoS with XML.) See the Notes: below for more information.

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

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>
```

Notes:

- If you do not set the **<participant_name>** property in the **<participant_qos>** settings, the *Record* tool will automatically build a participant name and set it using the prefix "**RTI Recorder:** ". This is for compatibility with *RTI Administration Console*. If this property is changed, the *Record* tool won't override the property, but compatibility between the *Record* tool and *RTI Administration Console* will be broken if the participant name is not prefixed with "**RTI Recorder:** " (notice the space after the colon).
- Transports are configured through the Property QoS under the **<participant_qos>** tag.

4.7.1 Enabling Monitoring Library in the Record Tool

RTI Monitoring Library enables *Connext DDS* 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).

4.7.2 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 5,050.

Therefore, if you have a data-type that would require more than 5,050 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];
}
```

4.8 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 4.9.)

Table 4.13, “TopicGroup Properties,” on page 4-28 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 4.13 **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
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

Table 4.13 **TopicGroup Properties**

Properties under <topic_group>	Syntax	Description
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 Connext DDS Core Libraries User's Manual* for more information. (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
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 4.14, "Topics Properties," on page 4-30.) If set to '*', everything is recorded. 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
index	<index> *User Data Table fields* </index>	Allows you to create an index for the recorded database based on the fields provided as parameters. See 'Create Index' Syntax (Section 4.8.1) for details.

Table 4.13 **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 **$** separator with the **path_separator** database property described in Table 4.4, "Database Properties," on page 4-10.) Thus, two topics with the same name but from two different TopicGroups are stored in separate tables. A shared table means that Topics with the same name are stored in the same table, regardless of where it was recorded from. In this case the table has an additional column, **table_prefix**, which stores the table prefix in the form: **RecordGroupName$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 4.14, "Topics Properties," on page 4-30.

Table 4.14 **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
		Note: Keep in mind that spaces are valid first characters in topic names, thus they can affect the matching process. For example, this will match both Triangle and Square topics (notice there is no space before Square):
		`<topic_expr>Triangle,Square</topic_expr>`
		However the following will only match Triangle topics (because there is a space before Square):
		`<topic_expr>Triangle, Square</topic_expr>`

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.

4.8.1 'Create Index' Syntax

SQLite indexes may improve performance on certain SQL queries. The *Replay* tool creates an index based on the reception timestamp when opening the database, if it did not already exist (see Performance and Indexing (Section 6.4)). The process of building the index, however, may take some time for large tables. But if it is important for you to avoid spending this time, or if indexing needs to happen during the recording, this is an easy way to create customized indexes while still recording data into the database.

Note: While other applications accessing the database may benefit from online indexing, the *Record* tool's performance may drop because of it, so you should consider this when using online indexing.

The *Record* tool's ability to create and store indexes for the User Data Table is controlled by the <index> property under <topic_group> (see Table 4.13, "TopicGroup Properties"). The <index> property expects one or more <field> tags, which create the indexes.

Syntax

The <topic_group> can contain as many indexes as needed (for some notes about indexes and performance see Indexing and Performance in SQLite: Tips and Tricks (Section 4.8.2)). Each of those indexes can be built using one or more fields. The fields listed could be meta-data fields (such as the reception timestamp used by the *Replay* tool) or user-data ones. In addition, you may optionally add a prefix to the index. The general syntax is:

```
<topic_group>
...
    <index prefix="OptionalPrefix">
        <field> One_field </field>
        <field> Another_field </field>
        ...
    </index>
...
</topic_group>
```

The above configuration will create an index named OptionalPrefix$TableName, where **$** is the default path-separator (although it could be replaced using the **<path_separator>** property) and **TableName** is the name of the user-data table (see Table 4.4, "Database Properties" for details).

There are some details to consider when creating an index:

- An index can only be created based on columns that actually exist in the user-data table. So if some or all of the columns specified in the index configuration are excluded (see Choosing Which SampleInfo and Discovery Fields to Record (Section 4.5.1)), they cannot be used to build an index.
- If the index cannot be created, a warning is shown and the *Record* tool will continue its execution.
- Two indexes cannot share the same prefix. If that happens, only the first one is created.
- If no prefix is provided, the string "**index**" and a ordinal number are added before the table name. For instance, if two indexes are created for a given table, their names will be **index1$TableName** and **index2$TableName**.

Examples:

To improve the *Replay* tool's initialization performance, we can create an index using the field **SampleInfo_reception_timestamp** like this:

```
<index>
    <field> SampleInfo_reception_timestamp </field>
</index>
```

Note that this is the same index that would be created by the *Replay* tool. (This index corresponds to the one created by the old database property **<create_index>**, which is no longer supported.)

User-data fields can also be used to create indexes. To add a user-data field to an index configuration, you must use the fully qualified name of the field. This notation is similar to the notation used to access data fields in filter expressions for content-filtered topics (see *RTI Connext DDS Core Libraries User's Manual* for more details). Fields inside container types are accessed using a period character '**.**' (e.g., **field1.nestedField2.nestedField3...**). Fields in collections (arrays and sequences) can be accessed using square brackets ('**[]**') notation (e.g., **collectionField[1].nestedField**, **matrixField[0][0].field1**). Only primitive fields can be used for indexing, because the *Record* tool does not store non-primitive fields.

For example, consider a race scenario. We have a collection of ten runners and their order in the race is recorded with the *Record* tool. The data types are defined using IDL as follows:

```
struct Runner {
    int id; //@key
    char * name; //@key
```

```
}
struct Race {
    struct Runner runnerList[10];
    TimeStamp currentTime;
}
```

If other applications are trying to access the recorded data in order to analyze it, we may want to speed up data retrieval for these applications by indexing the above data in many ways. For instance, we could build an index based on the ID and name of the first runner. We may also need an index based on the current time and the reception timestamp. The XML configuration to create these indexes is:

```
<topic_group>
...
    <!-- This is the index for the first runner -->
    <index prefix="firstRunnerIndex">
        <field> runnerList[0].id </field>
        <field> runnerList[0].name </field>
    </index>

    <!-- This is the index for the timestamps -->
    <index>
        <field> SampleInfo_reception_timestamp </field>
        <field> currentTime </field>
    </index>
...
</topic_group>
```

The above configuration creates two indexes:

- **firstRunnerIndex$TableName**, which indexes on the first runner's **id** and **name**.
- **index1$TableName**, which indexes on the **SampleInfo_reception_timestamp** field and user-data **currentTime** field.

4.8.2 Indexing and Performance in SQLite: Tips and Tricks

Online indexing can affect the permanence of the insertions the *Record* tool makes in the database, because of the extra work to maintain the indexes. However, in situations where more than one application besides the *Record* tool will be accessing the database, online indexing may make a difference, especially for applications retrieving data from the database.

It is important that indexes and the applications using them are as efficient as possible. Here are some tips and tricks about indexing and efficient usage of indexes in SQLite.

4.8.2.1 Small Size Data Fields Work Best as Index Columns

The fewer bytes used in every index entry, the better for performance. SQLite stores indexes as B-Trees. Every tree node uses a page of the database file. If the index does not fit in one page, overflow pages are created. Keeping the index in one page may make a difference in terms of performance. Therefore, large BLOBS, compact octet or char arrays, etc. should be avoided as index columns. If you need to index on one of these types, consider building a hash key or similar compact representation and use that for indexing instead. The space overhead may be worth the performance improvement.

4.8.2.2 Use Multi-Column Indexes where Possible

SQLite can use indexes that are created by indexing more than one column. However, SQLite only allows a single index per table within a simple SQL statement. For UPDATE and DELETE statements, this translates into only allowing a single index, since these statements can only work with one table at a time.

Suppose we have a table with columns C 1, C 2, ..., C n and we want to access the table with a SELECT statement with columns C 1, C 2, ..., C i, i < n, in the WHERE clause, such as "... WHERE C 1 = value 1 AND C 2 = value 2 ... AND C i = value i." SQLite will be able to use the index provided for each of the terms in the WHERE clause. However, if we have individual indexes for each column, SQLite can only access one of the indexes for the SELECT statement and will only benefit from the use of indexes for one of the search terms.

4.8.2.3 Avoid Excessive Indexes

Each index added to the database has a performance cost:

- Every additional index takes up some additional space in the database.
- INSERT and UPDATE statements have to modify both the original table and every index associated with it. The performance of INSERT and UPDATE decreases linearly with the number of indexes.
- Compilation of the SQL statements takes longer when there are more indexes for the compiler to choose from.
- More indexes to choose from, bigger the chance the optimizer selects a suboptimal index for a query.

Avoid redundant indexes where possible. If for a certain table we have index 1 indexing on columns (x,y) and index 2 indexing on columns (x,y,z) then index 1 is redundant and can be eliminated.

4.8.2.4 Use Indexes to Improve Performance of ORDER BY Clauses

To evaluate statements containing ORDER BY classes, SQLite first puts all the results of the SELECT statement in a temporary table, then sorts the temporary table according to the ORDER BY clause columns. This can be time-consuming. Whenever possible, SQLite tries to avoid storing and then sorting the result set; it does this by using an index that makes the results of the SELECT statement appear in order. To force this to happen, provide the table with an index that indexes all the columns in the ORDER BY clause.

For example, suppose we have a table with columns (x,y,z,a) and we want to execute the following query:

```
SELECT a FROM table ORDER BY x,y,z
```

Then providing an index in columns (x,y,z) will make SQLite avoid using a temporary table and sorting all the result set.

The index can also be used to satisfy a WHERE clause and an ORDER BY clause in the same statement. In the example above, suppose we want to execute this statement:

```
SELECT a FROM table WHERE x=value ORDER BY y,z
```

The above index on columns (x,y,z) will still be used by SQLite both to speed up the WHERE clause and to avoid sorting the whole result set in the ORDER BY statement.

Nevertheless, care must be taken so that the columns in the index are all used in the SQL statement. For example, consider the following statement:

```
SELECT a FROM table WHERE x=value ORDER BY z
```

It does not use "y", which creates a gap in the index terms of our index. This will make the index be used to speed up the WHERE clause, but won't avoid storing and sorting the whole result set afterwards.

4.8.2.5 Terms Used in Inequality Expressions should be Placed Last in Index Column Lists

SQLite allows at most one index column to be used in statements containing inequality expressions (such as '<', '>', '<=' or '>='). Columns that are constrained by inequalities should be placed as the right-most

term of the index. For example, suppose if we have a table with columns (x,y,z,a), an index in columns (x,y,z) and the following SQL statement:

```
SELECT a FROM table WHERE x = value x AND y < value y AND z = value z
```

The index does not help optimize term "z", because "y" is constrained by an inequality.

SQLite allows up to two inequality expressions for the same column, as long as they provide upper and lower bounds for the column. So this statement will effectively use the index as the inequality expressions for "y" provide an upper and lower bound for the term:

```
SELECT a FROM table WHERE x = value x AND y < value y1 AND y > value y2
```

4.8.2.6 Use Indexed Column Names in WHERE Clauses, not Expressions

SQLite uses indexes for a column when the column name appears isolated in statements, not in more complex expressions. For example, if we have a table with column "x" and an index in "x", the index will be used if the SQL statement includes "x" but not expressions based on "x", such as these:

```
... WHERE x - value = 0
... WHERE x * 1 = ?
... WHERE +x = value
```

All the WHERE clauses above will cause a full-table scan and SQLite will not use the index to optimize the query.

4.9 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 4.10) to switch to a different configuration file (one that does have a RecordGroup) and start recording.

Table 4.15 describes the RecordGroup properties. Notice that domain_ref and topic_ref are required.

Table 4.15 **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
subscriber_qos	<subscriber_qos> *DDS_SubscriberQos* </subscriber_qos>	Configures the Subscriber used by the RecordGroup. Default: default Subscriber QoS settings See the *RTI Connext DDS Core Libraries User's Manual* for details. (See the chapter on Configuring QoS with XML.)

Table 4.15 **RecordGroup Properties**

Properties under <record_group>	Syntax	Description
topic_ref	<topic_ref> *StringSequence* </topic_ref>	**Required.** Specifies a sequence of references to TopicGroups. Default: Null
user_topic_metadata_fields	<user_topic_metadata_fields> <included> <field> *field expression* </field> ... </included> <excluded> <field> *field expression* </field> ... </excluded> </user_topic_metadata_fields>	Specifies lists of included and excluded Sample Info or metadata fields in the user topic tables. A field expression is a POSIX fnmatch expression to match field names as described in Appendix A (e.g. "SampleInfo_*" will match all Sample Info specific fields). There is no upper limit in the number of <field> expressions to be specified. The 'included' and 'excluded' fields are optional. If both are defined, the 'included' expressions are processed before the 'excluded' ones. User topic field settings can also be expressed in the Database settings (see Table 4.4). However, Record Group settings take precedence over general database settings defined there.

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.)

4.10 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 *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

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

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.

You can learn more in the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

- *Recording Service* can subscribe to topics associated with optional, final, mutable, 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 and mutability (described in the *RTI Connext DDS Core Libraries 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 and mutable 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>
                                <register_top_level>false</register_top_level>
                                <type_name>ShapeTypeExtended</type_name>
                                <registered_type_name>
                                   <element>ShapeType</element>
                                </registered_type_name>
                                <topics>
                                   <element>Circle</element>
                                   <element>Square</element>
                                   <element>Triangle</element>
                                </topics>
                             </element>
```

```
                        </type>
                     </element>
                  </file_group>
                  <path>
                     <element>.</element>
                  </path>
               </xml>
            </type_config>
         </element>
      </domain_group>
   </domain_type_config>
```

With the above type configuration, *Recording Service* will register the type "ShapeTypeExtended" with the DomainParticipant associated with “domain0”. When any of the topics in the <topics> list is matched ("Circle", "Square" or "Triangle"), *Recording Service* will use "ShapeTypeExtended" as the type for recording it. The name used for registering the type depends on the <register_top_level> flag. If this flag is set to true, the type name used for the registration will be the canonical type name (in the example above, it would be "ShapeTypeExtended"). However, if this flag is set to false, at least one name has to be provided for registration of the type using the <registered_type_name> XML setting. The first name in this list will be the one used by *Recording Service*.

When *Recording Service* discovers the first DataWriter on topic “T”, it will first try to match the topic name against the <topics> expressions in all the <type> settings. If it finds a match, it will use the associated type settings. Otherwise, *Recording Service* will try to create the topic using the DataWriter’s registered type name. If the type name has not been registered with *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.

4.10.1.1 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 def -->
                       <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>
                                      <topics>
                                         <element>Square</element>
                                         <element>Circle</element>
                                         <element>Triangle</element>
                                      </topics>
```

```
                                    <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>
```

4.10.2 Recording Two Versions of a Type in Different Tables in Same Database

The following example shows how *Recording Service* can store samples of different type versions it finds for a topic "T" for the same DDS domain ID, in different tables, and without any loss or duplication of information (each type version is recorded completely, and no samples in one table are present in a different one).

The type configuration settings in *Recording Service* allow the specification of name filters (regular POSIX fn-match expressions) which allow the specification of a subset of the Recording Service domains they will apply to. This allows the specification of different type selection settings (as described in Selecting a Type Version For a Topic "T" In a Recording Domain (Section 4.10.1)) for different *Recording Service* domains.

With these tools and using *RTI Shapes Demo* as an example, we could define a domain where to record the base type of Shapes (named "domain0Base") and another one to use for the extended type ("domain0Extended") associated with the same domain ID:

```
<domain name="domain0Base">
    <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>

<domain name="domain0Extended">
    <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>
```

By applying filters to the Domain Type Configuration settings, we can specify that domain "domain0Base" should record topics "Circle", "Square" and "Triangle" with the base type in the XML definitions, like this:

```
<domain_type_config>
    <domain_group>
        <element>
            <domain_filter>
                <element> domain0Base </element>
            </domain_filter>
            <type_config>
                <xml>
                    <file_group>
                       <element>
                          <register_top_level>false</register_top_level>
                          <file_name>
                             <element>TestTypesLibrary.xml</element>
                          </file_name>
                          <type>
                             <element>
                                <register_top_level>true</register_top_level>
                                <type_name>ShapeType</type_name>
                                <topics>
                                   <element>Circle</element>
                                   <element>Square</element>
                                   <element>Triangle</element>
                                </topics>
                             </element>
                          </type>
                       </element>
                    </file_group>
                    <path><element>.</element></path>
                </xml>
            </type_config>
        </element>
```

The same way, we can associate the extended type for shapes with the extended domain:

```
            <element>
                <domain_filter>
                    <element> domain0Extended </element>
                </domain_filter>
                <type_config>
                    <xml>
                        <file_group>
                           <element>
                              <register_top_level>false</register_top_level>
                              <file_name>
                                 <element>TestTypesLibrary.xml</element>
                              </file_name>
                              <type>
                                 <element>
                                    <register_top_level>false</register_top_level>
                                    <type_name>ShapeTypeExtended</type_name>
                                    <registered_type_name>
                                       <element>ShapeType</element>
                                    </registered_type_name>
                                    <topics>
                                       <element>Circle</element>
                                       <element>Square</element>
                                       <element>Triangle</element>
                                    </topics>
                                 </element>
                              </type>
                           </element>
                        </file_group>
                        <path><element>.</element></path>
                    </xml>
                </type_config>
            </element>
        </domain_group>
    </domain_type_config>
```

For the moment, we have defined the type to use with each *Recording Service* domain, but because the default value for the TypeConsistencyEnforcementQosPolicy is ALLOW_TYPE_COERCION and both types are compatible, we would record each sample twice: once for each type version. We need to set the DataReader QoS settings in the TopicGroups we create so that they will not allow type coercion:

```
    <topic_group name="Shapes">
        <topics>
            <topic_expr>Circle,Square,Triangle</topic_expr>
        </topics>
        <field_expr> * </field_expr>
        <datareader_qos>
            <type_consistency>
                <kind>DISALLOW_TYPE_COERCION</kind>
            </type_consistency>
        </datareader_qos>
    </topic_group>
```

We also need to associate the RecordGroup we create with both *Recording Service* domains:

```
    <record_group name="RecordGroup">
        <domain_ref>
            <element>domain0Base</element>
            <element>domain0Extended</element>
        </domain_ref>
```

```
        <topic_ref>
            <element>Shapes</element>
        </topic_ref>
    </record_group>
```

With all these settings together, *Recording Service* will create a different table for each type version of Shapes it finds when it discovers each of the *Shapes Demo* topics. For example, for topic "Square", it would create table "Square$RecordGroup$domain0Base" where it would record only the samples published with the base type version, and table "Square$RecordGroup$domain0Extended", which would contain only the samples published with the extended type version of Shapes.

Note: Because of a known issue[1], for the *Replay* tool to work properly with the databases created with the above settings (replaying different topics with their recorded version requires concurrent access to the database from different Replay database entitites), we need to enable the <create_index> flag in *Recording Service* so the index is created beforehand:

```
<recorder_database>
    <database_name> shapes.dat </database_name>
    <create_index> true </create_index>
</recorder_database>
```

1. RTI Issue ID RECORD-318

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 DDS* 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 Recording Service Integration with Extensible Types (Section 4.10).
2. Create a *Connext DDS* 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 request" Topic (named RTI_RECORDER_COMMAND_REQUEST_TOPIC) and a DataWriter for "command response and status" Topic (named RTI_RECORDER_COMMAND_RESPONSE_TOPIC). So the *Record* tool will write status updates and comman responses.

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_RECORDER_COMMAND_TYPE** Register a type of this name, as seen in Figure 5.1.
- **RTI_RECORDER_COMMAND_REQUEST_TOPIC** Create a DataWriter with this Topic name, as seen in Figure 5.2.
- **RTI_RECORDER_COMMAND_RESPONSE_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_RECORDER_COMMAND_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 and command responses)
- 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 DDS* is assumed.)

Figure 5.2 **Creating the Required Entities**

```
struct DDS_TopicQos tqos = DDS_TopicQos_INITIALIZER;
DDS_Topic * dds_topic_cmd = NULL;
DDS_Topic * dds_topic_status = NULL;
struct DDS_DataWriterQos wqos = DDS_DataWriterQos_INITIALIZER;
RTIRecorderAdminMessageDataWriter * dds_writer = NULL;
struct DDS_DataReaderQos rqos = DDS_DataReaderQos_INITIALIZER;
RTIRecorderAdminMessageDataReader * dds_reader = NULL;

...

dds_topic_cmd =
           DDS_DomainParticipant_create_topic(
                  dds_participant, RTI_RECORDER_COMMAND_REQUEST_TOPIC,
                  RTI_RECORDER_COMMAND_TYPE, &tqos,
                  NULL, DDS_STATUS_MASK_NONE);
dds_topic_status =
           DDS_DomainParticipant_create_topic(
                  dds_participant, RTI_RECORDER_COMMAND_RESPONSE_TOPIC,
                  RTI_RECORDER_COMMAND_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 RTIRecorderAdminMessage {
    long destination_mask;
    RTIRemoteAdminAddress destination;
    long msg_id;
    RTIRecorderAdminUnion msg;
}; //@Extensibility MUTABLE_EXTENSIBILITY
```

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)**

```
RTIRecorderAdminMessage * msg = NULL;

...

msg = RTIRecorderAdminMessagePlugin_create_sample();
/* Handle creation errors */
if (msg == NULL) {
    ...
}
msg->msg._d = RTI_RECORDER_ADMIN_START;
```

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

```
RTIRecorderAdminMessage * msg = NULL;
DDS_ReturnCode_t retcode;
struct DDS_SampleInfo info;

msg =  RTIRemoteCtxMsgPlugin_create_sample();
/* Handle creation errors */
if (msg == NULL) {
    ...
}
msg->msg._d = RTI_RECORDER_ADMIN_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,
        RTI_RECORDER_CONFIG_MAX_STRING);

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

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

/* check for errors here ... */
while (no response) {
    retcode = RTIRecorderAdminMessageDataWriter_read_next_sample(
            dds_reader, msg, &info, &DDS_HANDLE_NIL);
    if (retcode != DDS_RETCODE_NO_DATA) {
        /* response received */
    }
    sleep(1);
}
```

5.3.1 Updating the Record Tool's Partition QoS Policy

For each Record Group in the XML configuration, *Recording Service* creates a DDS Subscriber. There is one Subscriber in each of the Record Group's domains. Sometimes it may be useful to change the Partition QoS settings for some of these Subscribers.

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

Direction	Message Type (add prefix: RTI_RECORDER_ADMIN)	Description
From: Your *Connext DDS* 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 begin recording if <auto_start_ mode> is set to RECORD_DISABLED. (See Table 4.2, "General Properties," on page 4-6). If <auto_start_ mode> is set to DISCOVERY_RECORD_ENABLED, it will only record discovery data. In these two cases, for the *Record* tool to start recording user-data, a remote START command is required. 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_PARTITION	Updates the Partition QoS in all or some of the *Record* tool's Record Groups (in their associated DDS Subscribers). See Updating the Record Tool's Partition QoS Policy (Section 5.3.1).
	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 DDS* 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.

For instance, suppose the *Record* tool needs to record data from producers that are organized into different partitions based on their geographical distribution (different locations are represented by different Partition strings). It is possible to specify a Partition QoS policy via XML configuration when the *Record* tool starts up. But suppose the data producers change location—and thus change their Partition QoS? The *Record* tool won't be able to keep track of the data without changing its Partition QoS too. (For details on the Partition QoS Policy, see the *RTI Connext DDS Core Libraries User's Manual*.)

You can change the Partition QoS for any or all Record Groups by using the RTI_RECORDER_ADMIN-_PARTITION message type. This command needs the following information:

- A collection of partition strings to be applied. As described in the *RTI Connext DDS Core Libraries User's Manual*, the partitions can be POSIX regular expressions. If no partition string is specified, the default partition (empty) is applied.
- A collection of Record Group names (or regular expressions matching Record Group names) for which the partitions will be applied. The partitions will be applied to all the DDS Subscribers created for the matching Record Groups. If no Record Group string is specified, the partitions are applied to all the Record Groups in the *Record* tool's configuration.

The partition update command is absolute, meaning that current partitions that are active for the matching Record Groups will be discarded and the new ones will be applied instead. For example, if a Record Group is already joined to partition "A", in order to have the Record Group join "A" and "B," the command will need to specify both "A" and "B" in their partition string collection.

Partition updates are applied per Record Group. If a Record Group is linked with multiple DomainParticipants, the Partition QoS policy will be updated for each Subscriber created for each of the Record Group's associated DomainParticipants. For example, consider the following configuration:

```
<recorder>
    ...
    <domain name="Domain0">
        <domain_id> 0 </domain_id>
    </domain>
    ...
    <domain name="Domain1">
        <domain_id> 1 </domain_id>
    </domain>
    ...
    <record_group name="RecordGroup">
        <domain_ref>
            <element> Domain0 </element>
            <element> Domain1 </element>
        </domain_ref>
        <topic_ref>
            <element> ... </element>
        </topic_ref>
    </record_group>
    ...
</recorder>
```

With the above configuration, there will be two DDS DomainParticipants, "Domain0" and "Domain1." Each of these DomainParticipants will contain a Subscriber associated with the Record Group called "RecordGroup." Sending a partition update command specifying "RecordGroup" will affect the Subscribers in both DomainParticipants. If you have a situation in which different partition changes need to be specified for each Subscriber, you should specify separate Record Groups in the *Record* tool's configuration. For the above example, this would mean splitting the Record Group in two, like this:

```
<recorder>
    ...
    <domain name="Domain0">
        <domain_id> 0 </domain_id>
    </domain>
    ...
    <domain name="Domain1">
        <domain_id> 1 </domain_id>
    </domain>
    ...
```

```
        <record_group name="RecordGroupDomain0">
            <domain_ref>
                <element> Domain0 </element>
            </domain_ref>
            <topic_ref>
                <element> ... </element>
            </topic_ref>
        </record_group>
        <record_group name="RecordGroupDomain1">
            <domain_ref>
                <element> Domain1 </element>
            </domain_ref>
            <topic_ref>
                <element> ... </element>
            </topic_ref>
        </record_group>
        ...
    </recorder>
```

This way, partition update commands can modify separate Partition QoS settings for each Record Group.

The Partition QoS Policy establishes limits on the length of the strings provided in the partition string collection. There can be up to 64 strings, with a maximum of 256 characters summed across all strings. When the *Record* tool receives a partition update command, it checks this limit by checking the length of all the partition strings in the command. If the summed length of the strings is larger than the limit, the partitions won't be applied and an error response will be sent back to the command issuer.

Note: For DDS Subscribers to match DDS Publishers, *either* of the two can use regular expressions as their partition strings, but not *both* at the same time. A regular expression partition has to be matched against a specific name partition string. It is the user's responsibility to ensure that at least one of the sides in the communication (Publisher or Subscriber) uses a pecific name partition string as the partition.

Any failure while processing the command will result in an error response from the *Record* tool to the command issuer. However, after validating the partition strings passed in the command, the processing is completed for all the Record Groups, even if an error happens while processing any of the QoS updates for any of them.

Depending on the status of the *Record* tool when the partition update command is issued, the following can happen:

- If the *Record* tool is running, the command will update the current DDS Subscribers associated with matching Record Groups. The same will happen when the *Record* tool is idle (paused).
- If the *Record* tool is stopped, the command will update the stored QoS settings for the Record Groups. Because the DDS subscriptions haven't been created yet when Recorder is stopped, the Partition QoS settings that were updated will take effect when the *Record* tool is started. The newly created DDS Subscribers will use the new Partition QoS settings as received in the partition update command.

Note: Partition updates are not instantaneous. Thus, the *Record* tool may lose samples while the partition changes take effect. This behavior is not different than that of any other *Connext DDS* application.

5.3.1.1 Sending a Partition Update Command to the Record Tool

The following is the IDL definition of the type used to specify the information in a Partition Update command:

```
const long PARTITION_MAX_LENGTH = 256;
const long MAX_PARTITIONS = 64;
const long MAX_RECORD_GROUP_EXPRESSIONS = 256;
const long RECORD_GROUP_EXPRESSION_MAX_LENGTH = 256;
```

```
struct RTIRecorderPartitionMessage {
    sequence<string<PARTITION_MAX_LENGTH>, MAX_PARTITIONS> partitions;
    sequence<string<RECORD_GROUP_EXPRESSION_MAX_LENGTH>,
        MAX_RECORD_GROUP_EXPRESSIONS> recordGroupExpressions;
};
```

A maximum of 64 (MAX_PARTITIONS) partition strings can be specified, with a maximum length of 256 characters (PARTITION_MAX_LENGTH). In a similar way, a maximum of 256 expressions to match Record Group names (MAX_RECORD_GROUP_EXPRESSIONS) can be specified, with a maximum length of 256 characters (RECORD_GROUP_EXPRESSION_MAX_LENGTH).

The following code snippet in C shows how to prepare and send a partition update command to the *Record* tool so that all the Record Groups in the configuration join partition "A":

```
RTIRemoteCtxMsgDataWriter * messageWriter = NULL;
RTIRemoteCtxMsg * message = NULL;
DDS_ReturnCode_t retcode;

/* Initialize the DDS entities used for the communication */
...

/* Create the message instance to be written to the tool */
message = RTIRemoteCtxMsgTypeSupport_create_data();
if (message == NULL) {
    /* Error handling */
}

/* Set the destination information for broadcast commands
 * (all zeroes)
 * Set the destination mask as empty too */
message->destination.app_id = 0;
message->destination.host_id = 0;
message->destination.instance_id = 0;
message->destination_mask = 0;

/* Set the type of the message to be a partition message */
message->msg._d = RTI_REMOTECTX_MSG_RECORDER_PARTITION;

/* Set the message ID; here we set it to zero for simplicity,
 * but this value should be auto-incremented with every new command
 * sent to the tool in order to correlate the commands with their
 * responses */
message->msg_id = 0;

/* To send a partition update command for all Record Groups in a
 * Recording configuration, set the collection of Record Group
 * names (expressions) to be empty */
if (!DDS_StringSeq_ensure_length(
           &message->msg._u.partitions.recordGroupExpressions, 0, 0)) {
    /* Error handling */
}

/* Ensure enough space to add the "A" partition string to the
 * collection of partitions*/
if (!DDS_StringSeq_ensure_length(
           &message->msg._u.partitions.partitions, 1, 1)) {
    /* Error handling */
}

/* Copy the string ("A") into the first string in the partition
```

```
 * string collection. We've used 'strncpy' in conjunction with the
 * maximum length of a partition string for correctness, even
 * though we know the length of string "A" fits the maximum length
 * of a partition string */
strncpy(DDS_StringSeq_get(&message->msg._u.partitions.partitions,
                          0), "A", PARTITION_MAX_LENGTH);

/* Write the message; recall that the 'accept_broadcast_commands'
 * must be enabled in the Record tool to be able to read this
 * command; if we know the destination information for the Record
 * tool (app id, host id and instance id), we can use this
 * information and don't have to enable 'accept_broadcast_commands'
 */
retcode = RTIRemoteCtxMsgDataWriter_write(
           RTIRemoteCtxMsg_writer, message,&DDS_HANDLE_NIL);
if (retcode != DDS_RETCODE_OK) {
    /* Error handling */
}
```

If we wanted to include a Record Group filter in our command, the filtering expression would have to be included in the collection of Record Group expressions. For example, to apply a partition update only to Record Groups ending with the suffix "Domain1," we could use the regular expression "***Domain1**" and use the following code (in C language):

```
RTIRemoteCtxMsgDataWriter * messageWriter = NULL;
RTIRemoteCtxMsg * message = NULL;
DDS_ReturnCode_t retcode;

/* Initialize the DDS entities used for the communication */
...
/* Create the message instance to be written to the tool */
message = RTIRemoteCtxMsgTypeSupport_create_data();
if (message == NULL) {
    /* Error handling */
}
...
/* To send a partition update command to Record Groups suffixed
 * with "Domain1," use the regular expression "*Domain1"; we need
 * to ensure enough space in the record group expressions
 * collection to hold one entry */
if (!DDS_StringSeq_ensure_length(
    &message->msg._u.partitions.recordGroupExpressions, 1, 1)) {
    /* Error handling */
}
/* Copy the string ("*Domain1") into the first string in the
 * record group expression string collection. We used 'strncpy' in
 * conjunction with the maximum length of a record group expression
 * string for correctness, even though we know the length of string
 * "*Domain1" fits the maximum length of a record group expression
 * string */
strncpy(DDS_StringSeq_get(
             &message->msg._u.partitions.recordGroupExpressions, 0),
        "*Domain1",
        RECORD_GROUP_EXPRESSION_MAX_LENGTH);

/* Set the length of the partition string collection to be zero;
 * The Record tool will set the default partition (empty) in the
 * matching Record Groups */
```

```
if (!DDS_StringSeq_ensure_length(
            &message->msg._u.partitions.partitions, 0,  0)) {
    /* Error handling */
}

/* Write the message */
retcode = RTIRemoteCtxMsgDataWriter_write(
                    RTIRemoteCtxMsg_writer, message, &DDS_HANDLE_NIL);
if (retcode != DDS_RETCODE_OK) {
    /* Error handling */
}
```

The example above uses an empty collection of partition strings. This means that the *Record* tool will apply the default partition (empty) to the Record Groups that match the regular expression "*Domain1" specified in the command.

5.4 Using the Example Remote-Access Application—Record Shell

The *Record Shell* is a *Connext DDS* 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.

To start the *Record Shell*, open a command prompt[1] change to the **<NDDSHOME>/bin** directory, and enter:

- On Linux and Mac OS X systems:

  ```
  rtirecsh -domain <domain ID>
  ```

- On Windows systems:

  ```
  rtirecsh.bat -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.

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

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

Command-line Option	Description
-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)
- pause (Section 5.4.1.6)
- resume (Section 5.4.1.7)
- shutdown (Section 5.4.1.8)
- start (Section 5.4.1.9)
- status (Section 5.4.1.10)
- stop (Section 5.4.1.11)

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_ mode> (see Table 4.2, "General Properties," on page 4-6) is set to RECORD_ENABLED, it is not necessary to run the **start** command to start the *Record* tool. If <auto_start_mode> is set to any other mode, 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
```

Where:

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

5.4.1.6 pause

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

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

```
pause <model>
```

5.4.1.7 resume

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

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

```
resume <model>
```

5.4.1.8 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.

5.4.1.9 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 fileset each time it is started.

5.4.1.10 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 ..........: 5.0.0
Timestamp ........: Mon Feb 18 20:02:48 2013
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
  - 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.

5.4.1.11 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 DDS* 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 **<NDDSHOME>/bin** directory. Then enter:

- On Linux and Mac OS X systems:

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

- On Windows systems:

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

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

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

The *Replay* tool is dynamically linked against the *Connext DDS* libraries. You should run the tool from the **<NDDSHOME>/bin** directory scripts—not from the executable files themselves. The scripts set all the paths and variables needed for the tool to find the shared libraries and run correctly.

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.
-heapSnapshotPeriod	Enables heap monitoring. *Recording Service* will generate a heap snapshot every <sec>. Default: heap monitoring is disabled.
-heapSnapshotDir	When heap monitoring is enabled, this parameter configures the directory where the snapshots will be stored. The snapshot filename format is RTI_<configurationName><processId><index>.log. Default: current working directory
-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.

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

Command-line Option	Description
-verbosity <value>	Specifies what type of logging information should be printed. Silent Exceptions (both *Connext DDS* and the *Replay* tool) Warnings (the *Replay* tool only) Information (the *Replay* tool only) Warnings (both *Connext DDS* and the *Replay* tool) Tracing (the *Replay* tool only) Tracing (both *Connext DDS* 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 DDS* (see Chapter 15 in the *RTI Connext DDS Core Libraries User's Manual*).

- **<NDDSHOME>/resource/xml/NDDS_QOS_PROFILES.xml**

  This file contains the *Connext DDS* 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.

The next locations are specific to *Replay*:

- **<NDDSHOME>/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_0_0 </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.
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: True

Table 7.1 **Replay Service Properties**

Property	Syntax	Description
use_original_partitions	<use_original_partitions> *DDS_Boolean* </use_original_partitions>	Controls whether or not the *Replay* tool should use the DDS Partition QoS information from the recorded discovery information data as it was originally received. For this feature to work, the recorded data must include some fields in the DCPSPublication table and the user topic table. For details on how to configure the *Record* tool to record this information, and more information about this feature, see Section 7.11 in the *RTI Recording Service User's Manual*. 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. Do not use both <filename> *and* <fileset> at the same time. Default: Undefined
fileset	<fileset> <path_prefix>*String*</path_prefix> <set_number>*DDS_Long*</set_number> <start_segment>*DDS_Long*</start_segment> <end_segment>*DDS_Long*</end_segment> </fileset>	Specifies a fileset to be used by the *Replay* tool. This fileset must contain the file prefix to be searched for, the set number, and the start and end segments. Do not use both <filename> *and* <fileset> at the same time. Note: The fileset replay feature is not compatible with the reposition command for timestamps outside of the current segment being replayed.

Table 7.2 **Replay Database Properties**

Properties under <replay_database>	Syntax	Description
participant	<participant> *Participant Properties* </participant>	See Table 7.3, "Participant Properties"
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. See Table 7.8, "Time Control Properties"
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 Connext DDS Core Libraries User's Manual's* chapter on Configuring QoS with XML. Defaults: See the *RTI Connext DDS* API Reference HTML documentation on DomainParticipants. See the Note: below for more information.

Note:

- If you do not set the **<participant_name>** property in the **<participant_qos>** settings, the *Replay* tool will automatically build a participant name and set it using the prefix "**RTI Replay:** ". This is for compatibility with *RTI Administration Console*. If this property is changed, the *Replay* tool won't override the property, but compatibility between the *Replay* tool and *RTI Administration Console* will be broken if the participant name is not prefixed with "**RTI Replay:** " (notice the space after the colon).

7.4.1 Enabling Monitoring Library with Replay

This section only applies if you want to use *RTI Monitoring Library* (included in *Connext DDS*), which enables *Connext DDS* 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 4.7.1). 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.

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 Connext DDS Core Libraries User's Manual's* chapter on Configuring QoS with XML. Defaults: See the *RTI Connext DDS* 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"

Table 7.4 **Session Properties**

Properties under <session>	Syntax	Description
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. See Table 7.8, "Time Control Properties"

7.6 Replay Topic Properties

Table 7.5 describes the Topics' properties.

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".
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. See Table 7.8, "Time Control Properties".

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.

Table 7.6 **Input Properties**

Properties under <input>	Syntax	Description
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.

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 Connext DDS Core Libraries 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.

7.6.1 Type Selection

This sections explains how the *Replay* tool obtains the type version (TypeCode) and the registered type name that will be used to replay a topic "T".

- First, if there are XML type-configuration settings (<type_config>) for the replay database (see Section 7.9) and the name of the topic to be replayed matches any of the topic name expressions provided using the <topics> tag within <type>, the *Replay* tool will use the XML definition of the type. The registered type name will be determined as follows:

  1. The type name explicitly defined by the user in <replay_topic><input><type_name>.
  2. If <type><register_top_level> is false, the name provided in <type><registered_type_name>
  3. The canonical name (fully qualified name in the XML file) of the type.

- **Second, if there is no XML type definition, the *Replay* tool will first try to get a candidate registered type name as follows:**

  1. The type name explicitly defined by the user in <replay_topic><input><type_name>.

2. If there is no explicit type name, the *Replay* tool will try to get the name using the column **type_name** in the DCPSPublication table. If the name is not unique, the *Replay* tool will report an error.

- **Third, if a valid registered type name was obtained, the *Replay* tool will try to get a type definition (TypeCode) associated with the given registered type name as follows:**

1. Using the type definitions under <type_config>
2. If there is no type definition available in XML, the tool will try to get the type definition using the **typecode** column in the DCPSPublication table. If there is no available or unique type definition, the *Replay* tool will report an error.

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>.

- 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**

Table 7.8 **Time Control Properties**

Properties under <time_control>	Syntax	Description
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
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 DDS* 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.

Chapter 8: Accessing the Replay Tool from a Remote Location 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.

7.8.1 Enabling RTI Distributed Logger in the Replay Tool

The *Replay* tool provides integrated support for *RTI Distributed Logger* (included in *Connext DDS*).

When you enable *Distributed Logger,* the *Replay* tool will publish its log messages to *Connext DDS*. Then you can use *RTI Monitor*[1] to visualize the log message data. Since the data is provided in a *Connext DDS* 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.

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

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 Connext DDS* 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 Connext DDS* 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 Connext DDS* 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 Connext DDS* API Reference HTML documentation on Publishers.
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 Connext DDS* API Reference HTML documentation on Subscribers.

```
<administration>
<name>Replay Service using Distributed Logger</name>
<domain_id>99</domain_id>
<distributed_logger>
    <enabled>true</enabled>
</distributed_logger>
</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* section of the *RTI Connext DDS Core Libraries User's Manual.*

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".
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.
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.
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 7.15 **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.
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
topics	<topics> <element> *Topic name string* </element> <topics>	This is a list of regular POSIX fn-name expressions (every <element> entry is equivalent to an expression). A topic whose name matches any of the expressions will be recorded using the type definition designated by <type_name>. If a topic matches more than one expression in different <type> entries, the first one that was matched will be used.
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.

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 RTI *Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

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

7.10.1 Selecting the Type Version to use when Replaying a Topic

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.

The type used by the *Replay* tool to replay a certain topic is obtained as described in Type Selection (Section 7.6.1). When a unique type name and type-code exist for a topic in the DCPSPublication table, the *Replay* tool will be able to use that information to replay it. In any other case, XML Type Configuration is needed. As an example, imagine that we have the database recorded in the example shown in Selecting a Type Version For a Topic “T” In a Recording Domain (Section 4.10.1). It is recorded using the extended type version of the *Shapes Demo* types. In order to replay the tables in the database as samples of the base type version, we could configure the *Replay* tool in the following manner:

```
<replay_service name="baseShapeTypeReplay">
<auto_exit>true</auto_exit>
<replay_database name="baseShapeTypeReplay_Database">
     <filename>shapes.dat_0_0</filename>
     <type_config>
           <xml>
           <file_group>
```

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

```
                        <element>
                        <register_top_level>false</register_top_level>
                        <file_name>
                               <element>ShapeType.xml</element>
                        </file_name>
                        <type>
                               <element>
                               <register_top_level>true</register_top_level>
                               <type_name>ShapeType</type_name>
                               <topics>
                                      <element>Circle</element>
                                      <element>Square</element>
                                      <element>Triangle</element>
                               </topics>
                               </element>
                        </type>
                        </element>
                 </file_group>
                 <path>
                        <element>.</element>
                 </path>
                 </xml>
          </type_config>
          <participant>
                 <domain_id>0</domain_id>
          </participant>
          <session name="baseShapeTypeReplay_Session">
          <replay_topic>
                 <input>
                 <topic_name>*</topic_name>
                 <domain_name>*</domain_name>
                 <record_group_name>*</record_group_name>
                 <type_name>*</type_name>
                 </input>
          </replay_topic>
          </session>
    </replay_database>
    </replay_service>
```

The example uses the same **ShapeType.xml** file defined in Selecting a Type Version For a Topic “T” In a Recording Domain (Section 4.10.1) and Recording Two Versions of a Type in Different Tables in Same Database (Section 4.10.2). By using the <topics> tag (see Table 7.15, “Type Registration Properties”) in the <type> settings in the XML Type Configuration, we make a direct mapping of the topics to the type to be used for replay. The above configuration will result in all *Shapes Demo* topics being replayed with the base ShapeType type definition.

7.10.2 Replaying Topics with Different Type Versions Stored in Different Tables

If the *Record* tool was set up to record each type version for a topic in its own table (see Recording Two Versions of a Type in Different Tables in Same Database (Section 4.10.2)), it seems interesting to be able to do the same with the *Replay* tool and have every type topic correctly replayed with the original type version. This can be done by using different Replay database entities configured to replay each of the recorded versions (distinguished by the name of the *Recording Service* domain that recorded them).

Consider the following Replay configuration:

```
<replay_service name="bothVersionsShapesReplay">
<auto_exit>true</auto_exit>
<replay_database name="bothVersionsShapesReplay_Database">
```

```
     <filename>shapes.dat_0_0</filename>
     <type_config>
     <xml>
            <file_group>
            <element>
                   <register_top_level>false</register_top_level>
                   <file_name>
                   <element>ShapeType.xml</element>
                   </file_name>
                   <type>
                   <element>
                          <register_top_level>true</register_top_level>
                          <type_name>ShapeType</type_name>
                          <topics>
                          <element>Circle</element>
                          <element>Square</element>
                          <element>Triangle</element>
                          </topics>
                   </element>
                   </type>
            </element>
            </file_group>
            <path>
            <element>.</element>
            </path>
     </xml>
     </type_config>
     <participant>
     <domain_id>0</domain_id>
     </participant>
     <session name="bothVersionsShapesReplay_Session">
     <replay_topic>
            <input>
            <topic_name>*</topic_name>
            <domain_name>domain0Base</domain_name>
            <record_group_name>*</record_group_name>
            <type_name>*</type_name>
            </input>
     </replay_topic>
     </session>
</replay_database>
<replay_database name="bothVersionsShapesReplay_DatabaseExtended">
     <filename>shapes.dat_0_0</filename>
     <type_config>
     <xml>
            <file_group>
            <element>
                   <register_top_level>false</register_top_level>
                   <file_name>
                   <element>ShapeType.xml</element>
                   </file_name>
                   <type>
                   <element>
                          <register_top_level>false</register_top_level>
                          <type_name>ShapeTypeExtended</type_name>
                          <registered_type_name>
                          <element>ShapeType</element>
                          </registered_type_name>
                          <topics>
```

```
                        <element>Circle</element>
                        <element>Square</element>
                        <element>Triangle</element>
                        </topics>
                    </element>
                    </type>
                </element>
                </file_group>
                <path>
                <element>.</element>
                </path>
            </xml>
            </type_config>
            <participant>
            <domain_id>0</domain_id>
            </participant>
            <session name="bothVersionsShapesReplay_SessionExtended">
            <replay_topic>
                <input>
                <topic_name>*</topic_name>
                <domain_name>domain0Extended</domain_name>
                <record_group_name>*</record_group_name>
                <type_name>*</type_name>
                </input>
            </replay_topic>
            </session>
    </replay_database>
    </replay_service>
```

In the above configuration, there are two Replay database entities, each replaying the base and the extended version of the topics, respectively. This is done by specifying the adequate domain name in the <replay_topic><input><domain_name> tags (one database replays domain "domain0Base" and the other one replays "domain0Extended"). For each of these database entities, different XML Type Configuration settings are provided. Topics "Circle", "Square" and "Triangle" are mapped to the type ShapeType in the base Database entity, and they are mapped to the type "ShapeTypeExtended" in the extended Database entity.

Note*:* There is a known issue when accessing the same database file from different Replay database entities when the indexes haven't been created yet[1]. There are three ways to avoid this issue:

1. Record with the <recorder_database><create_index> flag set to true so that the indexes are created by *Recording Service* while recording.
2. Set the <replay_database><readonly> to true. This will disable write-mode access to the database and thus indexes won't be created. However, this can affect the performance while replaying the data (see Performance and Indexing (Section 6.4)).
3. Index the database tables offline by using any SQLite access tool (for an example, see Chapter 9: Viewing Recorded Data) the indexes on the table can be created with the following SQL query (all on one line):

```
CREATE INDEX IF NOT EXISTS
    "index$[table name]" ON "[table name]"(reception_timestamp);
```

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

1. RTI Issue ID RECORD-318

7.11 Using the Recorded System's Original Partition QoS Information

Starting with version 5.3.0, the *Replay* tool can replay data in the DDS partition from which it was originally received. This feature is turned off by default. When the <use_original_partitions> tag in the configuration is set to true, *Replay* will scan the DCPSPublication table to look for all the DDS Publishers that Recorder discovered and will create DDS Publisher representing each of them, using the original Partition QoS. *Replay* then will create a DDS Writer for each publication found, and using the correct DDS Publisher for it. Replay will also detect and apply changes in the Partition QoS that the *Record* tool detected and recorded, in a timely manner.

7.11.1 Configuring Recorder to be compatible with Replay using original partition QoS

To work correctly, this feature needs certain fields that were recorded in the database to be replayed. These fields are:

- **In the DCPSPublication table (the built-in publication data discovery table):**
  - The reception timestamp: SampleInfo_reception_timestamp
  - The publication handle keys: PublicationData_key_0, _1, _2 and _3
  - The publisher handle keys: PublicationData_publisher_key_0, _1, _2 and _3
  - The partition QoS information: PublicationData_partition
  - The *Record* tool can be easily configured to record them by using the field selection mechanisms:

```
<builtin_topic_metadata_fields>
    <DCPSPublication_topic>
        <included>
            <field>SampleInfo_reception_timestamp</field>
            <field>PublicationData_key_*</field>
            <field>PublicationData_publisher_key_*</field>
            <field>PublicationData_partition</field>
        </included>
    </DCPSPublication_topic>
</builtin_topic_metadata_fields>
```

- **In the user topic tables:**
  - The publication handle: SampleInfo_publication_handle
  - Recorder can be configured to include it this way:

```
<user_topic_metadata_fields>
    <included>
        <field>SampleInfo_publication_handle</field>
    </included>
</user_topic_metadata_fields>
```

7.11.2 Compatibility with Other Features and Limitations

The compatibility of *Replay* with other features will be complete when there are no partition changes. *Replay's* writers will load the initial partition to be used and will use it throughout the execution of the tool.

If partition QoS changes happen (the Partition QoS is mutable), *Replay* has to schedule these changes according to the reception timestamp in the DCPSPublication table. As a general rule, this scheduling

works at the service and database level. It won't work well with features that apply to sessions or topics. This scheduling is compatible with other features:

- Starting, stopping, pausing, resuming the service: *Replay* will start, stop, pause, and resume partition change scheduling. However, *Replay* won't work correctly with the reposition (goto) command.
- Replaying multiple files: *Replay* will get all partition changes in all the segments and schedule them seamlessly.
- Looping: *Replay* will work correctly when the LOOP mode is set at the service or database level. *Replay's* behavior will be undetermined if the start mode is specific to sessions or topics.
- Replay rate: *Replay* will schedule partition changes based on the service or the database's replay rate. However, *Replay* will have undetermined behavior if specific rates are applied to sessions or topics.
- Replay start and stop times: When specifying a sub-range of data to be specified, via the start and stop time, *Replay* will work fine when this is specified at the service or database level. However, when start and/or stop times are specified at the session or topic level, *Replay's* behavior is undetermined.

7.11.3 QoS Considerations when Working with Partitions

The Partition QoS can be seen as a partition in the system's DDS topology, not in the system's data. This is, when we apply a partition to a DDS Publisher, we're affecting the way endpoint discovery will happen, but not the way the data will be received, which remains the same, based on the same QoS settings.

Thus, partition swapping is non-deterministic regarding time considerations, just like discovery. The same way, when *Replay* applies a partition change to its DDS Publishers, the partition change will happen at a non-deterministic moment in the future, after new discovery has finished. This should be taken into account when designing the system's QoS settings.

Other important aspects to consider are the interaction with other QoS settings, like Durability or Reliability. Because the Partition QoS is not a partition on the data like, for example, content-filtering is, after a partition change, a late joiner may receive old samples based on the writer's Reliability and Durability settings (like *Replay's*). This means *Replay* can try to emulate the data flow based on the partitions that it reads from the DCPSPublication table, although this flow will be affected by other QoS settings of Replay and the DDS topology Replay is writing into.

Chapter 8 Accessing the Replay Tool from a Remote Location

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

To start the Replay Shell:

Open a command prompt[1], change to the **<NDDSHOME>/bin** directory, and enter:

- On Linux and Mac OS X systems:

  ```
  rtireplaysh [options]
  ```

- On Windows systems:

  ```
  rtireplaysh.bat [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
-verbosity <value>	Specifies what type of logging information should be printed. Silent Exceptions (both *Connext DDS* and the *Replay* tool) Warnings (the *Replay* tool only) Information (the *Replay* tool only) Warnings (both *Connext DDS* and the *Replay* tool) Tracing (the *Replay* tool only) Tracing (both *Connext DDS* and the *Replay* tool) Default: 1

The *Replay Shell* commands use this format:

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

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

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 7-12)
  - 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.

Table 8.2 **Replay Shell's Commands**

Command	Description
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

The data can be viewed with any command-line or GUI-based tool capable of reading SQLite databases.

Chapter 10 Converting and Exporting Recorded Data

Recording Service includes a *Converter* tool that enables serialized or deserialized data recorded with the *Record* tool to be exported to CSV, HTML or XML formats. It also allows you to convert serialized data tables into deserialized data tables so that the recorded data can be easily explored via SQL/SQLite in a human-readable way.

To work with the database files created by the *Record* tool, *Converter* needs the **replay_compatibility** flag to be turned on (see Table 4.2), otherwise conversion is not guaranteed to work properly.

Tables that were recorded with field filters (field_expr) cannot be converted.

General Usage

The script to launch *Converter* is in **<NDDSHOME>/bin/**. To start the tool, open a command prompt, change to **<NDDSHOME>/bin**, and enter:

- On Linux and Max OS X systems:

  ```
  rtirecconv [options] fileset|filename
  ```

- On Windows systems:

  ```
  rtirecconv.bat [options] fileset|filename
  ```

The parameters **fileset** or **filename** represent the name of a recorded file set such as **recorded_db.dat_0** or a concrete file segment such as **recorded_db.dat_0_0**.

10.1 Exporting Data

Converter can export data recorded by the *Record* tool or *Recording Console* in three different formats: CSV (comma-separated values), XML (default export format if not specified) and HTML.

Even if the exported table is serialized, the output format of the data is deserialized. By default, only valid samples (data for which the **valid_data** Sample Info value was true) are exported; Sample Info columns are not exported.

Here is an example of how a samples from *RTI Shapes Demo* is exported by *Converter* into XML format (external XML formatting was applied after the conversion):

```
<rticonverter fileset="rti_recorder_default.dat_0">
    <Square_RecordAll_domain0
     registered_type_name="ShapeType" topic_name="Square">
        <sample  topic_name="Square" sample_nr="0">
            <Timestamp> 1381746283759122998 </Timestamp>
            <color length="7" max_length="0"> MAGENTA </color>
```

```
            <x> 134 </x>
            <y> 140 </y>
            <shapesize> 30 </shapesize>
            <fillKind> 1 </fillKind>
            <angle> 0.000000 </angle>
        </sample>
        <sample  topic_name="Square" sample_nr="1">
            ...
    </Square_RecordAll_domain0>
</rticonverter>
```

There are several options available to select how and what should be exported:

Flag	Values	Description
-format	[xml | csv | html]	Specifies what format to use to export the data. If this flag is not specified, XML format is used.
-includeInfo		If present in the call to *Converter*, the Sample Info fields will be exported with each sample.
-includeNonData		If present in the call to *Converter*, samples will be exported even if the associated Sample Info **valid_data** property is false.
-compact	[auto | yes | no]	The *Record* tool may store arrays and sequences of single-byte primitive types (characters and octets) in compact mode. This means that the array or sequence is stored in one only blob column in the recorded database. *Converter* can detect how data is stored in this regard and follow the same pattern when exporting the data. The **auto** mode (default) will detect the compactness of single-byte collections and use it for exporting the data. There are ways to force the converted data to be compact or non-compact. By specifying **yes**, the single-byte collections will be exported in compact form. By specifying **no**, the data will be exported in non-compact form.
-decodeChar	[hex | text]	This flag is used to specify how to export values of character type in arrays or sequences. The default format is 'text'. Explanation of the two formats: **Text**: If the character being exported is printable, it's printed directly. Otherwise, it's printed as two hex digits preceded by the escape character '\' (e.g. '**\0B**'). If exporting to XML or HTML, special/reserved characters like '>', '<', etc. are exported as their equivalent, '**&gt**', '**&lt**' and so forth. **Hex**: Every character is exported as two hex digits. Every exported character is separated by a dash. For example, **0B-FA-18-D3**... and so on.
-decodeOctet	[hex | text]	This flag is used to specify how to export byte values in arrays or sequences. **Text**: If the byte being exported is printable, it's printed directly as a character would be printed. Otherwise, it's printed as two hex digits preceded by the escape character '\' (e.g. **\0B**). If exporting to XML or HTML, special/reserved characters like '>', '<', etc. are exported as their equivalent, '**&gt**', '**&lt**' and so forth. **Hex** (default): every character is exported as two hex digits. Every exported character is separated by a dash. For example, **0B-FA-18-D3** represents four bytes.

Flag	Values	Description
-time	[epoch | gmt]	Every sample that is exported includes a "Timestamp" field which with the reception timestamp of the sample. This timestamp can be exported in two ways: **epoch**: The timestamp in nanoseconds since January 1 1970. **gmt**: The timestamp in GMT time format. For example: Fri Oct 11 17:25:03 2013.

10.1.1 Notes on Exporting to CSV

There is a widely used standard for the CSV format, RFC-4180, called "Common Format and MIME Type for CSV Files" (see https://tools.ietf.org/html/rfc4180). *Converter* aims to follow this standard. Thus, for string fields being exported that contain commas, double quotes or new line characters, *Converter* will wrap those strings in double quotes.

Most tools capable of importing CSV files work correctly with RFC-4180-compliant CSV files. We have tested LibreOffice® Calc and Microsoft® Excel®; both work fine. A note on new Microsoft Excel versions: to correctly import the CSV file, use the data source functionality. Thus, from the ribbon, select the Data tab, then **New Query, From File, From CSV**, and follow the wizard (we use the default character - a comma - to separate fields).

10.2 Deserializing Serialized Tables

Samples stored in serialized format are not human-readable. Exporting the data automatically deserializes serialized data but it may be the case that we need to keep the data as a SQLite database for inspection or data handling, or because we still want to be able to use the *Replay* tool. *Converter* provides an option to process a SQLite database and transform all serialized tables into deserialized tables: **-deserialize**. This option takes precedence over any other format option, like HTML or XML. When **-deserialize** is included the format specifier is ignored.

By default, the new database file is created in the same location as the original one and the new file is prefixed with "deserialized." followed by the original file's name. The two files, original and deserialized, are equivalent. If the *Replay* tool was able to work with the original file, it will be able to replay the deserialized file.

10.3 Handling Data Types

Converter infers the type-code that should be used to deserialize and/or export the data from the DCPSPublication table and the topic name. However, the type-code may be missing, either because it wasn't published (not propagated by DomainParticipants during discovery) or because it was filtered by using field selection properties. It may also be desirable to force types to be loaded using XML configuration, e.g. in cases where different versions of the types were published and we want fine-grained control of the type being exported.

If the type-code is not directly accessible from the DCPSPublication table, *Converter* can provide XML representation of the types via the Recorder XML configuration file and name. The following command-line parameters can be used to provide *Converter* with XML type information:

Flag	Parameters	Description
-typeConfig	<XML configuration file> <configuration name>	When this option is present and a correct Recorder XML configuration file and configuration name are provided (see note below), the XML type configuration settings in the configuration (see Recording Service Integration with Extensible Types (Section 4.10) will be used to match the topics and types in the database and provide type-codes for them. **Important Note:** When we say "correct Recorder XML configuration file" we mean the actual Recorder configuration used to record the data, which includes a <domain_type_config> tag. If this tag was not included while recording (see Automatic Mode and the Use of XML Types (Section 4.6.1)), add it now on top of the Recorder configuration. But don't create a Recorder configuration that only contains that tag because *Converter* will not be able to use it correctly.
-forceXMLTypes		This option forces *Converter* to obtain the type-codes from the XML configuration provided via the **-typeConfig** option, whenever possible.

10.4 Selecting Output Files

Converter extracts the file set from the input file (or set) provided by the user and detects all available segments for conversion. Conversion is performed on every segment, and there is only one output file, which aggregates all the conversion results. So, for example, if we have file segments **rti_record_default.dat_0_0** and **rti_record_default.dat_0_1**, running *Converter* on any of the files or the file set (**rti_record_default.dat_0**) will convert both segments into the same file.

Note: It is not possible to convert a single segment with *Converter*. If there is more than one recorded file in a fileset *Converter* will automatically detect it and convert all of them.

The output file is created by appending the appropriate file extension depending on the conversion format. For example, if we are converting the fileset mentioned above into XML format, the output file would be called **rti_record_default.dat_0.xml**, located in the same directory as the original fileset.

You can, however, change the output file. The command-line options to alter the output file are as follows:

Flag	Parameters	Description
-outputFile	<filename>	If this option is present and has a valid file name parameter, the conversion results will be placed in the file passed as a parameter instead of the default filename derived from the converted fileset's name.
-filePrefix	<prefix>	This option has a string parameter used as a prefix for each of the output files created. This option cannot be used at the same time as **-outputFile**. When this option is present, an output file is created for each table in the fileset, prefixed with the prefix provided by the user. For example, if a recording contains tables Table1 and Table2 and the **-filePrefix** option is used with prefix "toXML", two output files will be created: **toXML_Table1.xml** and **toXML_Table2.xml**.

10.5 Exporting Discovery Tables

Converter's default behavior is to export just the user topic tables, except when deserializing, where the discovery tables are copied automatically to the deserialized SQLite database. It is possible to tell *Converter* to export the discovery tables when converting. If the **-includeDiscovery** option is present, *Converter* will export the discovery tables present in the recorded file alongside the user data tables.

10.6 Filtering User Topic Tables

Converter provides the ability to filter the tables to be exported. If the **-tableExpr <expression>** command-line option is used, *Converter* will export or convert tables that follow the patterns described by this option. The expression parameter for this option is a POSIX fnmatch expression. The **tableExpr** option can be repeated. If the name of a table matches any of the expressions provided with **tableExpr**, it is converted/exported; otherwise it isn't.

Discovery tables are not affected by the **tableExpr** option. If **-includeDiscovery** is included, the discovery tables will be exported normally. The **-deserialize** option will automatically include the discovery tables in the output database file, no matter what table expression is used. However, user topic tables will be filtered according to the table expression.

Chapter 11 Example Configuration Files

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

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 type-codes are available from the system.

Configuration File

```
<dds>
<recorder name=”scenario1”>
    <recorder_database>
           <database_name> mydomaindata </database_name>
           <max_file_size> 1 GB </max_file_size>
    </recorder_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”>
    <recorder_database>
            <database_name> mydomaindata </database_name>
            <max_file_size> 1 GB </max_file_size>
    </recorder_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">
    <recorder_database>
           <database_name> mydomaindata </database_name>
           <max_file_size> 2000 kB </max_file_size>
           <max_file_segments> 1000 </max_file_segments>
           <rollover> yes </rollover>
    </recorder_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_DESERIALIZEMODE_NEVER</deserial-
ize_mode>
    </domain>
...
</recorder>
</dds>
```

Expected Outcome

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

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.

Appendix A Fields Available for Recording

This appendix lists the fields that are available for the *Record* tool to store in both user topic tables and discovery tables. See Section 4.5.1 to learn how to include or exclude fields from the recorded tables.

A.1 User Topic Tables

The fields in Table A.1 are available for storing with the user data samples being recorded in user topic tables. Fields coming from the SampleInfo objects are prefixed with "SampleInfo_". More information on the meaning of the fields can be found in the *RTI Connext DDS* API Reference HTML documentation. The units of the timestamp fields are nanoseconds.

The *Record* tool stores two metadata fields with the user topic tables: the domain ID and the table prefix (see the **shared_table** property in Table 4.13 for more information on table prefixes).

User data fields are stored after column **SampleInfo_valid_data**. When recording in serialized format, the column storing the sample is called **rti_serialized_sample**. The sample's endianness and length are stored in columns **rti_serialized_endian** and **rti_serialized_length**, respectively.

Table A.1 **Fields Available to Store in User Topic Tables**

Field Name	SQL Type
SampleInfo_reception_timestamp	INTEGER
SampleInfo_source_timestamp	INTEGER
SampleInfo_valid_data	INTEGER
SampleInfo_publication_seq_nr	INTEGER
SampleInfo_subscription_seq_nr	INTEGER
SampleInfo_sample_state	INTEGER
SampleInfo_instance_state	INTEGER
SampleInfo_instance_handle	BLOB
SampleInfo_publication_handle	BLOB
SampleInfo_disposed_generation_count	INTEGER
SampleInfo_no_writers_generation_count	INTEGER
SampleInfo_sample_rank	INTEGER
SampleInfo_generation_rank	INTEGER
SampleInfo_absolute_generation_rank	INTEGER

Table A.1 **Fields Available to Store in User Topic Tables**

Field Name	SQL Type
SampleInfo_original_publication_virtual_guid	BLOB
SampleInfo_original_publication_virtual_seq_nr	INTEGER
SampleInfo_related_original_publication_virtual_guid	BLOB
SampleInfo_flag	INTEGER
SampleInfo_source_guid	BLOB
SampleInfo_related_source_guid	BLOB
SampleInfo_related_subscription_guid	BLOB
SampleInfo_related_original_publication_virtual_seq_nr	INTEGER
Metadata_domain_id	INTEGER
Metadata_table_prefix	TEXT

A.2 DCPSParticipant Table (Discovery)

The DCPSParticipant table stores the samples of the built-in discovery topic DCPSParticipant samples received by the *Record* tool. Samples of this topic are prefixed with "ParticipantData_". Alongside every sample of the discovery topic, the Sample Info object received is also stored (and prefixed with "SampleInfo_"). For more information on this topic, see the *RTI Connext DDS Core Libraries* API Reference HTML documentation. The *Record* tool stores two metadata fields with this table: the domain ID and the name of the Recorder Domain object as defined in the recorder configuration that generated the discovery traffic.

Table A.2 **Fields Available to Store in DCPSParticipant Tables**

Field Name	SQL Type
SampleInfo_reception_timestamp	INTEGER
SampleInfo_source_timestamp	INTEGER
SampleInfo_valid_data	INTEGER
ParticipantData_key_0	INTEGER
ParticipantData_key_1	INTEGER
ParticipantData_key_2	INTEGER
ParticipantData_key_3	INTEGER
ParticipantData_userdata	BLOB
ParticipantData_participant_name	TEXT
ParticipantData_property	TEXT
ParticipantData_rtps_protocol_version	TEXT
ParticipantData_rtps_vendor_id	BLOB
ParticipantData_dds_builtin_endpoints	INTEGER
ParticipantData_default_unicast_locators	TEXT
ParticipantData_product_version	TEXT
SampleInfo_publication_seq_nr	INTEGER
SampleInfo_subscription_seq_nr	INTEGER

Table A.2 **Fields Available to Store in DCPSParticipant Tables**

Field Name	SQL Type
SampleInfo_sample_state	INTEGER
SampleInfo_instance_state	INTEGER
SampleInfo_instance_handle	BLOB
SampleInfo_publication_handle	BLOB
SampleInfo_disposed_generation_count	INTEGER
SampleInfo_no_writers_generation_count	INTEGER
SampleInfo_sample_rank	INTEGER
SampleInfo_generation_rank	INTEGER
SampleInfo_absolute_generation_rank	INTEGER
SampleInfo_original_publication_virtual_guid	BLOB
SampleInfo_original_publication_virtual_seq_nr	INTEGER
SampleInfo_related_original_publication_virtual_guid	BLOB
SampleInfo_related_original_publication_virtual_seq_nr	INTEGER
SampleInfo_flag	INTEGER
SampleInfo_source_guid	BLOB
SampleInfo_related_source_guid	BLOB
SampleInfo_related_subscription_guid	BLOB
Metadata_domain_id	INTEGER
Metadata_domain_name	TEXT

A.3 DCPSPublication Table (Discovery)

This table stores the samples of the built-in discovery topic DCPSPublication samples that are received by Recorder. Samples of this topic are prefixed with "PublicationData_". Alongside every sample of the discovery topic, the Sample Info object received is also stored (and prefixed with "SampleInfo_"). For more information on this topic, see the *RTI Connext DDS Core Libraries* API Reference HTML documentation. The *Record* tool stores two metadata fields with this table: the domain ID and the name of the Recorder Domain object as defined in the recorder configuration that generated the discovery traffic.

Table A.3 **Fields Available to Store in DCPSPublication Tables**

Field Name	SQL Type
SampleInfo_reception_timestamp	INTEGER
SampleInfo_source_timestamp	INTEGER
SampleInfo_valid_data	INTEGER
PublicationData_typecode_length	INTEGER
PublicationData_topic_name	TEXT
PublicationData_type_name	TEXT
PublicationData_typecode	BLOB
PublicationData_key_0	INTEGER
PublicationData_key_1	INTEGER

Table A.3 **Fields Available to Store in DCPSPublication Tables**

Field Name	SQL Type
PublicationData_key_2	INTEGER
PublicationData_key_3	INTEGER
PublicationData_participant_key_0	INTEGER
PublicationData_participant_key_1	INTEGER
PublicationData_participant_key_2	INTEGER
PublicationData_participant_key_3	INTEGER
PublicationData_topic_data	BLOB
PublicationData_destination_order_kind	INTEGER
PublicationData_destination_order_scope	INTEGER
PublicationData_destination_order_source_timestamp_tolerance	INTEGER
PublicationData_presentation_access_scope	INTEGER
PublicationData_presentation_coherent_access	INTEGER
PublicationData_presentation_ordered_access	INTEGER
PublicationData_partition	TEXT
PublicationData_group_data	BLOB
PublicationData_publisher_key_0	INTEGER
PublicationData_publisher_key_1	INTEGER
PublicationData_publisher_key_2	INTEGER
PublicationData_publisher_key_3	INTEGER
PublicationData_durability_kind	INTEGER
PublicationData_durability_direct_communication	INTEGER
PublicationData_deadline_period	INTEGER
PublicationData_latency_budget_duration	INTEGER
PublicationData_liveliness_kind	INTEGER
PublicationData_liveliness_lease_duration	INTEGER
PublicationData_reliability_kind	INTEGER
PublicationData_reliability_acknowledgment_kind	INTEGER
PublicationData_reliability_max_blocking_time	INTEGER
PublicationData_ownership_kind	INTEGER
PublicationData_ownership_strength	INTEGER
PublicationData_user_data	BLOB
PublicationData_property	BLOB
PublicationData_virtual_guid	BLOB
PublicationData_rtps_protocol_version	TEXT
PublicationData_rtps_vendor_id	BLOB
PublicationData_product_version	TEXT
PublicationData_disable_positive_acks	INTEGER
PublicationData_locator_filter	TEXT
PublicationData_lifespan_duration	INTEGER
PublicationData_durability_service_service_cleanup_delay	INTEGER
PublicationData_durability_service_history_kind	INTEGER

Table A.3 **Fields Available to Store in DCPSPublication Tables**

Field Name	SQL Type
PublicationData_durability_service_history_depth	INTEGER
PublicationData_durability_service_max_samples	INTEGER
PublicationData_durability_service_max_instances	INTEGER
PublicationData_durability_service_max_samples_per_instance	INTEGER
PublicationData_unicast_locators	TEXT
PublicationData_publication_name	TEXT
PublicationData_publication_role_name	TEXT
SampleInfo_publication_seq_nr	INTEGER
SampleInfo_subscription_seq_nr	INTEGER
SampleInfo_sample_state	INTEGER
SampleInfo_instance_state	INTEGER
SampleInfo_instance_handle	BLOB
SampleInfo_publication_handle	BLOB
SampleInfo_disposed_generation_count	INTEGER
SampleInfo_no_writers_generation_count	INTEGER
SampleInfo_sample_rank	INTEGER
SampleInfo_generation_rank	INTEGER
SampleInfo_absolute_generation_rank	INTEGER
SampleInfo_original_publication_virtual_guid	BLOB
SampleInfo_original_publication_virtual_seq_nr	INTEGER
SampleInfo_related_original_publication_virtual_guid	BLOB
SampleInfo_related_original_publication_virtual_seq_nr	INTEGER
SampleInfo_flag	INTEGER
SampleInfo_source_guid	BLOB
SampleInfo_related_source_guid	BLOB
SampleInfo_related_subscription_guid	BLOB
Metadata_domain_id	INTEGER
Metadata_domain_name	TEXT

A.4 DCPSSubscription Table (Discovery)

This table stores the samples of the built-in discovery topic DCPSSubscription samples that are received by the *Record* tool. Samples of this topic are prefixed with "SubscriptionData_". Alongside every sample of the discovery topic, the Sample Info object received is also stored (and prefixed with "SampleInfo_"). For more information on this topic, see the *RTI Connext DDS Core Libraries* API Reference HTML documentation. The *Record* tool stores two metadata fields with this table: the domain ID and the name of the Recorder Domain object as defined in the recorder configuration that generated the discovery traffic.

Table A.4 **Fields Available to Store in DCPSSubscription Tables**

Field Name	SQL Type
SampleInfo_reception_timestamp	INTEGER
SampleInfo_source_timestamp	INTEGER
SampleInfo_valid_data	INTEGER
SubscriptionData_key_0	INTEGER
SubscriptionData_key_1	INTEGER
SubscriptionData_key_2	INTEGER
SubscriptionData_key_3	INTEGER
SubscriptionData_participant_key_0	INTEGER
SubscriptionData_participant_key_1	INTEGER
SubscriptionData_participant_key_2	INTEGER
SubscriptionData_participant_key_3	INTEGER
SubscriptionData_topic_name	TEXT
SubscriptionData_type_name	TEXT
SubscriptionData_destination_order_kind	INTEGER
SubscriptionData_destination_order_scope	INTEGER
SubscriptionData_destination_order_source_timestamp_tolerance	INTEGER
SubscriptionData_presentation_access_scope	INTEGER
SubscriptionData_presentation_coherent_access	INTEGER
SubscriptionData_presentation_ordered_access	INTEGER
SubscriptionData_partition	TEXT
SubscriptionData_group_data	BLOB
SubscriptionData_topic_data	BLOB
SubscriptionData_subscriber_key_0	INTEGER
SubscriptionData_subscriber_key_1	INTEGER
SubscriptionData_subscriber_key_2	INTEGER
SubscriptionData_subscriber_key_3	INTEGER
SubscriptionData_durability_kind	INTEGER
SubscriptionData_durability_direct_communication	INTEGER
SubscriptionData_deadline_period	INTEGER
SubscriptionData_latency_budget_duration	INTEGER
SubscriptionData_liveliness_kind	INTEGER
SubscriptionData_liveliness_lease_duration	INTEGER
SubscriptionData_reliability_kind	INTEGER
SubscriptionData_reliability_acknowledgment_kind	INTEGER
SubscriptionData_reliability_max_blocking_time	INTEGER
SubscriptionData_ownership_kind	INTEGER
SubscriptionData_user_data	BLOB
SubscriptionData_property	BLOB
SubscriptionData_unicast_locators	TEXT
SubscriptionData_multicast_locators	TEXT
SubscriptionData_virtual_guid	BLOB

Table A.4 **Fields Available to Store in DCPSSubscription Tables**

Field Name	SQL Type
SubscriptionData_rtps_protocol_version	TEXT
SubscriptionData_rtps_vendor_id	BLOB
SubscriptionData_product_version	TEXT
SubscriptionData_disable_positive_acks	INTEGER
SubscriptionData_time_based_filter_minimum_separation	INTEGER
SubscriptionData_content_filter_property	TEXT
SubscriptionData_subscription_name	TEXT
SubscriptionData_subscription_role_name	TEXT
SampleInfo_publication_seq_nr	INTEGER
SampleInfo_subscription_seq_nr	INTEGER
SampleInfo_sample_state	INTEGER
SampleInfo_instance_state	INTEGER
SampleInfo_instance_handle	BLOB
SampleInfo_publication_handle	BLOB
SampleInfo_disposed_generation_count	INTEGER
SampleInfo_no_writers_generation_count	INTEGER
SampleInfo_sample_rank	INTEGER
SampleInfo_generation_rank	INTEGER
SampleInfo_absolute_generation_rank	INTEGER
SampleInfo_original_publication_virtual_guid	BLOB
SampleInfo_original_publication_virtual_seq_nr	INTEGER
SampleInfo_related_original_publication_virtual_guid	BLOB
SampleInfo_related_original_publication_virtual_seq_nr	INTEGER
SampleInfo_flag	INTEGER
SampleInfo_source_guid	BLOB
SampleInfo_related_source_guid	BLOB
SampleInfo_related_subscription_guid	BLOB
Metadata_domain_id	INTEGER
Metadata_domain_name	TEXT