RTI Recording Service

User’s Manual

Version 6.0.0
# Contents

1 Introduction ................................. 2
    1.1 Introduction .................................. 2
    1.2 The Basics .................................. 2
    1.3 Paths Mentioned in Documentation ............... 3

2 Installation .................................. 5

3 Recording Service ......................... 6
    3.1 Usage ..................................... 6
        3.1.1 Starting Recording Service ................. 6
        3.1.2 Stopping Recording Service ................. 6
        3.1.3 Recording Service Command-Line Parameters .... 6
    3.2 Configuration ............................. 8
        3.2.1 How to Load the XML Configuration ............ 8
        3.2.2 XML Syntax and Validation .................... 9
        3.2.3 Builtin Configuration of Recording Service .... 10
        3.2.4 XML Tags for Configuring Recording Service .... 10
        3.2.5 Recording Service Tag ...................... 11
            Example: Specify a Recording Service Configuration in XML 12
        3.2.6 Administration ........................... 15
        3.2.7 Monitoring ............................... 15
        3.2.8 Storage ................................ 16
        3.2.9 SQLite .................................. 16
        3.2.10 Fileset .................................. 18
        3.2.11 Rollover .................................. 21
        3.2.12 Plugin .................................. 22
        3.2.13 DomainParticipant ......................... 22
        3.2.14 Session .................................. 23
        3.2.15 Topic Group .............................. 25
        3.2.16 Topic ..................................... 25
            Example: Configuration to enable both FlatData and zero-copy transfer over shared memory 27
        3.2.17 Enabling Distributed Logger .................. 28
        3.2.18 Recording Service Builtin Configuration Details .... 29
    3.3 Remote Administration ..................... 30
        3.3.1 Enabling Remote Administration ............... 30
        3.3.2 Available Service Resources .................. 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3 Remote API Overview</td>
<td>31</td>
</tr>
<tr>
<td>3.3.4 Recording Service</td>
<td>32</td>
</tr>
<tr>
<td>3.3.5 Storage</td>
<td>33</td>
</tr>
<tr>
<td>3.4 Monitoring</td>
<td>34</td>
</tr>
<tr>
<td>3.4.1 Overview</td>
<td>34</td>
</tr>
<tr>
<td>Enabling Service Monitoring</td>
<td>34</td>
</tr>
<tr>
<td>Monitoring Types</td>
<td>34</td>
</tr>
<tr>
<td>3.4.2 Monitoring Metrics Reference</td>
<td>35</td>
</tr>
<tr>
<td>Service</td>
<td>35</td>
</tr>
<tr>
<td>Session</td>
<td>38</td>
</tr>
<tr>
<td>TopicGroup</td>
<td>39</td>
</tr>
<tr>
<td>Topic</td>
<td>40</td>
</tr>
<tr>
<td>3.5 Tutorials</td>
<td>41</td>
</tr>
<tr>
<td>3.5.1 Getting Started with Recording Service and Shapes Demo</td>
<td>41</td>
</tr>
<tr>
<td>Edit the Configuration</td>
<td>41</td>
</tr>
<tr>
<td>Start Shapes Demo</td>
<td>42</td>
</tr>
<tr>
<td>Start Recording Service</td>
<td>42</td>
</tr>
<tr>
<td>View the Data in Sqlite3</td>
<td>44</td>
</tr>
<tr>
<td>3.5.2 Using Recording Service and Admin Console</td>
<td>44</td>
</tr>
<tr>
<td>Configuration</td>
<td>44</td>
</tr>
<tr>
<td>Start Recording Service</td>
<td>45</td>
</tr>
<tr>
<td>Start Shapes Demo</td>
<td>45</td>
</tr>
<tr>
<td>Viewing with Admin Console</td>
<td>45</td>
</tr>
<tr>
<td>Administering with Admin Console</td>
<td>49</td>
</tr>
<tr>
<td>3.5.3 Using Recording Service as a Library</td>
<td>49</td>
</tr>
<tr>
<td>Include files</td>
<td>49</td>
</tr>
<tr>
<td>Using the RecordingService class</td>
<td>50</td>
</tr>
<tr>
<td>3.5.4 Plugging in Custom Storage</td>
<td>50</td>
</tr>
<tr>
<td>Custom Storage API Overview</td>
<td>51</td>
</tr>
<tr>
<td>3.5.5 Accessing JSON samples through SQL</td>
<td>52</td>
</tr>
<tr>
<td>3.5.6 Controlling Recording Service Remotely from an Application</td>
<td>52</td>
</tr>
<tr>
<td>3.5.7 Listing the Timestamp Tags in a Recording</td>
<td>53</td>
</tr>
<tr>
<td>3.6 Troubleshooting</td>
<td>53</td>
</tr>
<tr>
<td>3.6.1 Verbosity</td>
<td>53</td>
</tr>
</tbody>
</table>

### 4 Replay Service

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Usage</td>
<td>55</td>
</tr>
<tr>
<td>4.1.1 Starting Replay Service</td>
<td>55</td>
</tr>
<tr>
<td>4.1.2 Stopping Replay Service</td>
<td>55</td>
</tr>
<tr>
<td>4.1.3 Replay Service Command-Line Parameters</td>
<td>55</td>
</tr>
<tr>
<td>4.1.4 Replay Service Runtime Behavior</td>
<td>57</td>
</tr>
<tr>
<td>4.2 Configuration</td>
<td>57</td>
</tr>
<tr>
<td>4.2.1 How to Load the XML Configuration</td>
<td>57</td>
</tr>
<tr>
<td>4.2.2 XML Syntax and Validation</td>
<td>59</td>
</tr>
<tr>
<td>4.2.3 XML Tags for Configuring Replay Service</td>
<td>59</td>
</tr>
<tr>
<td>4.2.4 Replay Service Tag</td>
<td>60</td>
</tr>
<tr>
<td>Example: Specify a Replay Service Configuration in XML</td>
<td>62</td>
</tr>
</tbody>
</table>
5.2 Converter Configuration ............................................................. 94
  5.2.1 How to Load the XML Configuration ........................................ 94
  5.2.2 XML Syntax and Validation .................................................. 96
  5.2.3 Built-in Configuration of Converter ........................................ 96
  5.2.4 XML Tags for Configuring Converter ....................................... 96
  5.2.5 Converter Tag .................................................................. 98
  Example: Specify a Configuration in XML ......................................... 99
  5.2.6 Input Storage ................................................................... 100
  5.2.7 Output Storage .................................................................. 100
  5.2.8 SQLite ............................................................................ 100
  5.2.9 Fileset ............................................................................ 102
  5.2.10 Rollover ......................................................................... 103
  5.2.11 Legacy ............................................................................ 104
  5.2.12 Domain Mapping .............................................................. 104
  5.2.13 Plugin ............................................................................ 105
  5.2.14 Data Selection ................................................................. 105
  5.2.15 Time Range .................................................................... 105
  5.2.16 DomainParticipant ........................................................... 106
  5.2.17 Session ........................................................................ 106
  5.2.18 Topic Group .................................................................... 107
  5.2.19 Topic ............................................................................ 107
  5.2.20 Converter’s Built-in Configuration Details ................................. 108
  5.3 Tutorials ............................................................................... 109
    5.3.1 Using Timestamp Tags with Converter .................................... 109

6 Common Infrastructure .................................................................. 111
  6.1 Application Resource Model ..................................................... 111
    6.1.1 Example: Simple Resource Model of a Connext DDS Application .. 112
      Resource Identifiers .............................................................. 112
    6.1.2 Example: Resource Identifiers of a Generic Connext DDS Application .. 113
  6.2 Remote Administration Platform ............................................. 113
    6.2.1 Remote Interface ............................................................. 114
      Standard Methods ............................................................... 115
      Custom Methods ............................................................... 115
    6.2.2 Communication ............................................................... 116
      Reply Sequence .................................................................. 118
      Example: Accessing from Connext DDS Application ...................... 119
    6.2.3 Common Operations ........................................................ 120
      Create Resource .................................................................. 120
      Get Resource ...................................................................... 121
      Update Resource .................................................................. 121
      Set Resource State ................................................................ 122
      Delete Resource .................................................................. 123
  6.3 Monitoring Distribution Platform .............................................. 123
    6.3.1 Distribution Topic Definition .............................................. 124
      Example: Monitoring of Generic Application ............................... 125
    6.3.2 DDS Entities .................................................................... 127
    6.3.3 Monitoring Metrics Publication ............................................ 127
7 Release Notes

7.1 Supported Platforms ......................................................... 134
7.2 Compatibility .................................................................. 134
7.3 What’s New in 6.0.0 .............................................................. 134
  7.3.1 Avoiding gaps in Replay when using time ranges ................. 135
7.4 What’s Fixed in 6.0.0 .............................................................. 135
  7.4.1 Pluggable storage for Converter discarded plugin properties specified in XML 135
7.5 Known Issues .................................................................. 135
  7.5.1 Recording Service may fail when current working directory in c:\Program Files 135

8 Copyrights ........................................................................ 137
Welcome to *RTI® Recording Service*, an *RTI Connext® DDS* application that records and replays DDS Topics and discovery data.
Chapter 1

Introduction

1.1 Introduction

*RTI Recording Service* includes the following tools:

- *Recording Service*, an *RTI Connext DDS* application that records Topics and discovery data. *Recording Service* records updates to data along with a timestamp, so you can view or replay updates to data in your system as they occur over time. Recorded data is stored in SQLite® files by default. *Recording Service* also has an API to record to a custom data store.

- *Replay Service*, an application that can play back data recorded by *Recording Service*. *Replay Service* also has an API to allow plugging in custom storage.

- *Converter*, an application that converts binary (serialized) recorded data to deserialized data that can be viewed and queried. The most efficient way to record data is in serialized form. *Converter* allows data to be recorded efficiently, then post-processed into a queryable form. *Converter* also provides APIs to plug in custom storage.

1.2 The Basics

*Recording Service* is used when you need to record updates to system data over a period of time, and to access that data by time. One example of when you might use the *Recording Service*: if you are testing your system, you can use *Recording Service* to record all DDS data updates that occur during a particular test run. Then you can use a database tool or *Replay Service* to view what happened at specific times during your test.

*Recording Service*’s built-in database is a SQLite file. *Recording Service* can store data in the SQLite database in two formats:

- CDR serialized format: This is the format in which data is sent over the network, so it is the most efficient way to store data. It is binary, so it is not queryable.

- JSON format: To store data this way, *Recording Service* must convert between the network format and the JSON format. This is queryable, but comes at a performance cost.

*Recording Service* also provides an API that allows you to implement your own storage backend. You will receive the data in serialized format, then use *Connext DDS* dynamic-data APIs to...
deserialize the data.

Figure 1.1: *Recording Service* receives DDS samples over the network and records them to storage, based on their timestamps.

*Replay Service* works with data recorded by *Recording Service*. It uses the timestamps of the recorded data to replay the data back into a DDS system at the original offsets between recorded timestamps. *Replay Service* also offers an API to retrieve your data from the storage of your choice to be replayed.

Converter is a tool that can convert between the data formats used by *Recording Service*. Currently, the only built-in formats supported by *Recording Service* are the CDR and JSON formats described above, within a SQLite database. Converter allows you to record data in the efficient CDR format and later convert to a queryable JSON format.

### 1.3 Paths Mentioned in Documentation

This 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`
Figure 1.3: *Converter* takes serialized (binary()) data and converts it to deserialized JSON data.

- 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

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

Whenever 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\rticlouddiscoveryservice.bat"

Or if you have defined the NDDSHOME environment variable: "%NDDSHOME%\bin\rticlouddiscoveryservice.bat"

- *<path to examples>* By default, 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 the copied examples as *<path to examples>*.

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

Default path to the examples:

- 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 10 systems, the folder is C:\Users\your user name\Documents.
Chapter 2

Installation

*RTI Recording Service* comes pre-installed as part of *RTI Connext DDS*. Contact [support@rti.com](mailto:support@rti.com) for information on how to obtain a package for your platform.
Chapter 3

Recording Service

3.1 Usage

This section explains how to run Recording Service from a command line. In particular, it describes:

- How to Start Recording Service (Section 3.1.1)
- How to Stop Recording Service (Section 3.1.2)
- Recording Service command-line parameters (Section 3.1.3)

3.1.1 Starting Recording Service

Recording Service runs as a separate application. The script to run the executable is in `<NDDSHOME>/bin`. (See Section 1.3 for the path to NDDSHOME.)

To start Recording Service with a default configuration, enter:

```
rtirecordingservice
```

This command will run Recording Service indefinitely until you stop it.

You can run Recording Service using the command-line parameters defined below.

**Note:** Recording Service is pre-loaded with a builtin configuration that has default settings. See Section 3.2.18.

3.1.2 Stopping Recording Service

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

3.1.3 Recording Service Command-Line Parameters

The following table describes all the command-line parameters available in Recording Service. To list the available parameters, run `rtirecordingservice -help`. 
All command-line parameters are optional; if specified, they override the values of any corresponding settings in the loaded XML configuration. See Section 3.2.4 for the XML elements that can be overridden with command-line parameters.

### Table 3.1: Recording Service Command-Line Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-appName &lt;string&gt;</td>
<td>Application name used to identify this execution for remote administration and to name the Connext DDS participant.</td>
</tr>
<tr>
<td>-cfgFile &lt;string&gt;</td>
<td>Semicolon-separated list of configuration file paths. Default: unspecified</td>
</tr>
<tr>
<td>-cfgName</td>
<td>Configuration name used to find a &lt;recording_service&gt; matching tag in the configuration file.</td>
</tr>
<tr>
<td>-domainIdBase &lt;int&gt;</td>
<td>This value is added to the domain IDs in the &lt;domain_participant&gt; tag in the configuration file. For example, if you set -domainIdBase to 50 and use domain IDs 0 and 1 in the configuration file, then Recording Service will use domains 50 and 51 for recording (though the recorded data will appear as though it was recorded in domains 0 and 1). Default: 0</td>
</tr>
<tr>
<td>-D&lt;name&gt;=&lt;value&gt;</td>
<td>Defines a variable that can be used as an alternate replacement for XML environment variables, specified in the form $(VAR_NAME). Note that definitions in the environment take precedence over these definitions.</td>
</tr>
<tr>
<td>-help</td>
<td>Shows this help.</td>
</tr>
<tr>
<td>-heapSnapshotDir</td>
<td>Output directory where the heap monitoring snapshots are dumped. The filename format is: RTI_heap_&lt;appName&gt;<em>&lt;processId&gt;</em>&lt;index&gt;.log</td>
</tr>
<tr>
<td>-heapSnapshotPeriod &lt;sec&gt;</td>
<td>Period at which heap monitoring snapshots are dumped. Enables heap monitoring if &gt; 0. Default: 0 (disabled)</td>
</tr>
<tr>
<td>-logFormat &lt;format&gt;</td>
<td>A mask to configure the format of the log messages for both Recording Service and Connext DDS:</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT - Print message, method name, and activity context</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPED - Print message, method name, activity context, and timestamp</td>
</tr>
<tr>
<td></td>
<td>• MINIMAL - Print only message number and method name</td>
</tr>
<tr>
<td></td>
<td>• MAXIMAL - Print all available fields</td>
</tr>
<tr>
<td></td>
<td>Default: DEFAULT</td>
</tr>
<tr>
<td>-maxObjectsPerThread &lt;int&gt;</td>
<td>Maximum number of thread-specific objects that can be created. Default: 1024.</td>
</tr>
<tr>
<td>-remoteAdministrationDomainId &lt;int&gt;</td>
<td>Enables remote administration and sets the domain ID for communication. Default: Remote administration is not enabled.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-remoteMonitoringDomainId &lt;int&gt;</td>
<td>Enables remote monitoring and sets the domain ID for status publication. <strong>Default:</strong> Remote monitoring is not enabled.</td>
</tr>
<tr>
<td>-verbosity [0-6]</td>
<td><strong>RTI Service verbosity</strong>&lt;br&gt;• 0 - silent&lt;br&gt;• 1 - exceptions (<em>Connext DDS</em> and <em>Recording Service</em>)&lt;br&gt;• 2 - warnings (<em>Recording Service</em>)&lt;br&gt;• 3 - information (<em>Recording Service</em>)&lt;br&gt;• 4 - warnings (<em>Connext DDS</em> and <em>Recording Service</em>)&lt;br&gt;• 5 - tracing (<em>Recording Service</em>)&lt;br&gt;• 6 - tracing (<em>Connext DDS</em> and <em>Recording Service</em>)&lt;br&gt;<strong>Default:</strong> 1 (exceptions)</td>
</tr>
<tr>
<td>-version</td>
<td>Prints the program version and exits.</td>
</tr>
</tbody>
</table>

### 3.2 Configuration

When you start *Recording Service*, you can specify a configuration file in XML format. In that file, you can set properties that control the behavior of the service. This section describes how to write a configuration file.

#### 3.2.1 How to Load the XML Configuration

*Recording Service* loads its XML configuration from multiple locations. Here are the various sources of configuration files, listed in load order:

- `[working directory]/USER_RECORDING_SERVICE.xml` This file is loaded automatically if it exists.
- `[NDDSHOME]/resource/xml/RTI_RECORDING_SERVICE.xml` This file is loaded automatically if it exists.
- File specified using the command-line parameter `-cfgFile`.

**Note:** `[working directory]` indicates the path to the current working directory from which you run *Recording Service*.

[NDDSHOME] indicates the path to your *Connext DDS* installation. See Section 1.3.

You may use a combination of the above sources and load multiple configuration files.

Here is an example configuration file. You will learn the meaning of each line as you read the rest of this section.
3.2.2 XML Syntax and Validation

The XML representation of DDS-related resources must follow these syntax rules:

- It shall be a well-formed XML document according to the criteria defined in clause 2.1 of the Extensible Markup Language standard.
- It shall use UTF-8 character encoding for XML elements and values.
- It shall use <dds> as the root tag of every document.

To validate the loaded configuration, Recording Service relies on an XSD file that describes the format of the XML content. We recommend including a reference to this document in the XML file.
that contains the service’s configuration; this document provides helpful features in code editors such as Visual Studio®, Eclipse®, and NetBeans®, including validation and auto-completion while you are editing the XML file.

The XSD definitions of the XML elements are in $NDDSHOME/resource/schema/rti_recording_service.xsd.

To include a reference to the XSD document in your XML file, use the attribute xsi:noNamespaceSchemaLocation in the <dds> tag. For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="[NDDSHOME]/resource/schema/rti_recording_service.xsd">
    <!-- ... -->
</dds>
```

*Note:* 
[NDDSHOME] indicates the path to your Connext DDS installation. See Section 1.3.

### 3.2.3 Builtin Configuration of Recording Service

Recording Service is pre-configured with a builtin configuration. See Section 3.2.18 for more details.

### 3.2.4 XML Tags for Configuring Recording Service

This section describes the XML tags you can use in a Recording Service configuration file. The following diagram and table describe the top-level tags allowed within the root <dds> tag.

**Table 3.2: Top-level Tags in Recording Service’s Configuration File**

<table>
<thead>
<tr>
<th>Tags within &lt;dds&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;qos_library&gt;</td>
<td>Specifies a QoS library and profiles. The contents of this tag are specified in the same manner as for the Connext DDS QoS profile file—see Configuring QoS with XML in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..*</td>
</tr>
<tr>
<td>&lt;types&gt;</td>
<td>Defines types that can be used by Recording Service. This is needed if data types are not available through discovery, or when using a transformation. The type description is done using the Connext DDS XML format for type definitions. See Creating User Data Types with Extensible Markup Language (XML), in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..*</td>
</tr>
</tbody>
</table>

Continued on next page
### 3.2.5 Recording Service Tag

A configuration file must have at least one `<recording_service>` tag. This tag is used to configure an execution of Recording Service.

A configuration file may contain multiple `<recording_service>` tags. When you start Recording Service, you can use the `-cfgName` command-line parameter to specify which `<recording_service>` tag to use to configure the service. This means one file can be used to configure multiple Recording Service executions.

The following diagram and Table 3.3 describe the tags allowed within a `<recording_service>` tag.

---

Table 3.3: Recording Service Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;recording_service&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;administration&gt;</code></td>
<td>Enables remote administration. When administration is enabled, monitoring is also enabled by default. If no domain ID is specified for monitoring, Recording Service will use the same domain as administration by default. See Section 3.2.6.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

---

Continued on next page
<table>
<thead>
<tr>
<th>Tags within &lt;recording_service&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;monitoring&gt;</td>
<td>Enables monitoring for the recording service, including statistics.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>See Section 3.2.7.</td>
<td></td>
</tr>
<tr>
<td>&lt;storage&gt;</td>
<td>Describes how the data will be stored. If this is not specified, data will be stored in a SQLite file using the default name “rti_recorder_default.dat”. See Section 3.2.8.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;domain_participant&gt;</td>
<td><strong>Required.</strong> Specifies a DomainParticipant to use to record data. Attributes</td>
<td>1..*</td>
</tr>
<tr>
<td></td>
<td>• name: Uniquely defines a DomainParticipant. <strong>Required.</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;domain_participant name=&quot;Participant3&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;domain_id&gt;3&lt;/domain_id&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;!-- Participant QoS --&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/domain_participant&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Section 3.2.13.</td>
<td></td>
</tr>
<tr>
<td>&lt;session&gt;</td>
<td><strong>Required.</strong> Active component of <em>Recording Service</em> for recording data. Contains one or more threads that can be used for recording. Attributes</td>
<td>1..*</td>
</tr>
<tr>
<td></td>
<td>• name: Uniquely defines a recording session. <strong>Required.</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• default_participant_ref: Specifies a default DomainParticipant to be used by children of this recording session. Children can override this by specifying their own participant.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;session name=&quot;Session&quot; default_participant_ref=&quot;Participant3&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;!-- ... topics / groups of topics to record --&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/session&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Section 3.2.14.</td>
<td></td>
</tr>
</tbody>
</table>

**Example:** Specify a Recording Service Configuration in XML

```xml
<dds>
  <recording_service name="MyRecorderService">
    <!-- ... Required entities -->
  </recording_service>
</dds>
```

Starting a *Recording Service* instance with the following command will use the `<recording_service>` tag with the name “MyRecorderService”:

```
$NDDSHOME/bin/rtirecordingservice -cfgFile file.xml -cfgName MyRecorderService
```
Figure 3.1: Top-level Tags in Recording Service's Configuration File
Figure 3.2: Tags used to configure a Recording Service instance
3.2.6 Administration

The `<administration>` tag allows you to enable and configure remote administration of Recording Service, including stopping, starting, and pausing recording.

See Section 3.3 for details on using remote administration.

<table>
<thead>
<tr>
<th>Tags within <code>&lt;administration&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;domain_id&gt;</code></td>
<td>Domain ID used for remote administration. Also used for monitoring by default.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;participant_qos&gt;</code></td>
<td>QoS used by the administration DomainParticipant. If the tag is not defined, Connext DDS defaults will be used.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;publisher_qos&gt;</code></td>
<td>QoS used by the administration Publisher. If the tag is not defined, Connext DDS defaults will be used.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;subscriber_qos&gt;</code></td>
<td>QoS used by the administration Subscriber. If the tag is not defined, Connext DDS defaults will be used.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| `<datawriter_qos>`            | QoS used by administration DataWriter(s). If the tag is not defined, Connext DDS defaults will be used, with the following changes:
   - history.kind = DDS_KEEP_ALL_HISTORY_QOS
   - resource_limits.max_samples = 32 | 0..1          |
| `<datareader_qos>`            | Quality of Service (QoS) used by administration DataReader(s). If the tag is not defined, the Connext DDS defaults will be used, with the following changes:
   - reliability.kind = DDS_RELIABLE_RELIABILITY_QOS (this value cannot be changed)
   - history.kind = DDS_KEEP_ALL_HISTORY_QOS
   - resource_limits.max_samples = 32 | 0..1          |
| `<distributed_logger>`        | When you enable Distributed Logger, Recording Service will publish its Log messages to Connext DDS. See Section 3.2.17. | 0..1          |

The contents of the tags for configuring QoS are specified in the same manner as for the Connext DDS QoS profile file. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.

3.2.7 Monitoring

The `<monitoring>` tag allows you to enable and configure remote monitoring of Recording Service. See Section 3.4.
Table 3.5: Monitoring Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;monitoring&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;enabled&gt;</td>
<td>Whether to enable monitoring of the service.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>Default: Disabled, unless administration is enabled.</td>
<td></td>
</tr>
<tr>
<td>&lt;domain_id&gt;</td>
<td>Domain ID used for monitoring.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>Default: The domain ID specified for monitoring.</td>
<td></td>
</tr>
<tr>
<td>&lt;datawriter_qos&gt;</td>
<td>QoS used by monitoring DataWriter(s)</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;publisher_qos&gt;</td>
<td>QoS used by monitoring Publisher(s)</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;participant_qos&gt;</td>
<td>QoS used by monitoring DomainParticipant</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;statistics_sampling_period&gt;</td>
<td>How frequently to sample the service’s statistics, using the tags &lt;sec&gt; or &lt;nanosec&gt;. For example, &lt;sec&gt;1&lt;/sec&gt; samples the service's statistics every second.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;status_publication_period&gt;</td>
<td>How frequently to publish the service status, using the tags &lt;sec&gt; or &lt;nanosec&gt;. For example, &lt;sec&gt;1&lt;/sec&gt; publishes the service’s status every second.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

The contents of the tags for configuring QoS are specified in the same manner as for the Connext DDS QoS profile file. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.

3.2.8 Storage

The <storage> tag allows you to configure the storage to which data will be written. You can choose between using the builtin SQLite storage or implementing your own storage plugin.

Table 3.6: Storage Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;storage&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sqlite&gt;</td>
<td>Enables storing data in a SQLite database file. See Section 3.2.9.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;plugin&gt;</td>
<td>Enables storing data in an external library that you specify. See Section 3.2.12 for more about using this tag. See Section 3.5.4 for a tutorial on plugging in custom storage.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

3.2.9 SQLite

The <sqlite> tag allows you to specify the name and file extension of a SQLite file in which to write data. It also allows you to choose the storage format in which to store the data. The default format, XCDR_AUTO, records data without deserializing it from the network format, so it is the most efficient way to store data; however, it is a binary format that cannot be queried without using Converter to convert it to a readable JSON_SQLITE format. The JSON_SQLITE format is slower to record, because it requires deserializing the data, but it can be queried using SQLite tools.

3.2. Configuration
Table 3.7: SQLite Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;sqlite&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;fileset&gt;</td>
<td>Set of files to write to, and parameters for creating files and directories in that set. See Section 3.2.10.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;file&gt;</td>
<td>File to write to. Default: rti_recorder_default</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;overwrite_policy_kind&gt;</td>
<td>Whether Recording Service is allowed to overwrite files. The options are OVERWRITE or DO_NOT_TOUCH. When DO_NOT_TOUCH is selected, Recording Service cannot overwrite an existing file, even if the rollover functionality is enabled. Default: OVERWRITE</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;storage_format&gt;</td>
<td>Specifies what format the data is stored in. The options are: - XCDR_AUTO: This is the binary format used by Connext DDS when sending data over the network. This has the highest performance for recording, but can only be viewed by using Converter to convert the data to a readable format, or by using Replay to replay the data. This will internally store data in XCDR or XCDR2 depending on the format received. - JSON_SQLITE: This format can be queried, but recording in this format has lower performance because data must be deserialized before it can be stored. - XCDR: The format to use when communicating with Connext DDS before 6.0.0. - XCDR2: More efficient than XCDR, used by Connext DDS 6.0.0 and later. Default: XCDR_AUTO</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;sql_initialization_string&gt;</td>
<td>Specifies a SQLite SQL expression to use when establishing sqlite connections using this plugin. This can be used to change the pragmas used by SQLite, or to do other database operations. <strong>Note:</strong> when using Recording Service and another application (either Replay Service or another SQLite application) at the same time to access the same database files, we recommend using SQLite’s WAL (write-ahead logging) mode. This can be done by adding PRAGMA JOURNAL_MODE = WAL; to this configuration setting. More information about SQLite’s WAL mode can be found here. Default: PRAGMA SYNCHRONOUS = OFF; PRAGMA JOURNAL_MODE = MEMORY;</td>
<td>0..1</td>
</tr>
</tbody>
</table>
3.2.10 Fileset

The `<fileset>` tag allows you to specify a set of files for *Recording Service* to write to. This lets you specify behaviors such as “create a new directory with each run of *Recording Service* based on the timestamp when the tool was started” or “create a new file every time *Recording Service* is started, incrementing an integer in the filename.”

The `<fileset>` tag is also where the rollover behavior is specified.

<table>
<thead>
<tr>
<th>Tags within <code>&lt;fileset&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;workspace_dir&gt;</code></td>
<td>Base directory where the database files for an instance of <em>Recording Service</em> (including discovery and user data files) will be stored. Depending on the value of the <code>&lt;execution_dir_expression&gt;</code> tag, this will either contain a set of files or a set of directories. Default: The current working directory.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
### Table 3.8 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;fieldset&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;execution_dir_expression&gt;</td>
<td>When <em>Recording Service</em> starts, it will use this expression to create the directory where output files will be stored. Every time <em>Recording Service</em> starts, it will evaluate this expression to decide on its output directory. Stopping and restarting remotely will cause <em>Recording Service</em> to re-evaluate this expression and possibly change its output directory. This execution directory is a parameterisable expression. In it, it accepts text and any combination of the following:</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>• Autonumeric. <strong>Format:</strong> %auto:M-N%. This parameter describes an integer that auto-increments every time <em>Recording Service</em> starts. The numeric sequence is restarted when <em>Recording Service</em> is manually shut down and restarted. M must be lower than N; together they define a numeric range, both inclusive. N can be omitted (%auto:M%), resulting in an unlimited sequence of numbers starting at M. <em>Example:</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;execution_dir_expression&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>test_run_%auto:0-3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/execution_dir_expression&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This example will create directories named test_run_0, test_run_1, test_run_2, and test_run_3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Timestamp. <strong>Format:</strong> %ts%. This parameter will take the current timestamp in the system (the time represented as number of seconds since Epoch).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Time. <strong>Format:</strong> %T%. Current time expressed in ISO 8601 time format (HH:MM:SS). Example: 14:55:02. This parameter uses the strftime() parameter %T.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Short date. <strong>Format:</strong> %F%. Short date in YYYY-MM-DD format. Example: 2001-08-23. This parameter uses the strftime() parameter %F.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Date and time. <strong>Format:</strong> %c%. Date and time representation, locale-dependent. This parameter uses the strftime() parameter %c.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Using no parameters will yield the same execution directory every time the service is started. In this case, if the directory already contains database files, they may be overwritten (see the <code>&lt;overwrite&gt;</code> tag). Default: %ts% (current timestamp number since Epoch).</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.8 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;file-set&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <filename_expression>          | Once Recording Service knows the exact directory in which to put the database files, it will use this parameter to determine the name(s) of the user data file(s) to be created. Right before the recording starts and every time Recording Service has to change its current file to a new one, it will use this parameterisable expression to generate the next file’s name. This setting accepts text and any combination of the following parameters:  
  • Autonumeric. **Format: %auto:M-N%.** This parameter describes an integer that auto-increments every time Recording Service is started. However, the numeric sequence is restarted with every execution of the Recording Service application. M must be lower than N; together M and N define a numeric range, both inclusive. N may be omitted (%auto:M%), resulting in an unlimited sequence of numbers starting at M. **Example:**
    
    ```xml
    <filename_expression>
    test_files_%auto:0-2%.dat
    </filename_expression>
    ```
    This will create files named test_files_0.dat, test_files_1.dat, and test_files_2.dat. When a rollover event occurs, Recording Service will either create one of these files, or overwrite the next file in the sequence. When Recording Service is restarted, this will start over with overwriting test_files_0.dat.  
  • Timestamp. **Format: %ts%.** This parameter will take the current timestamp in the system (the time represented as number of seconds since Epoch).  
  • Time. **Format: %T%.** Current time expressed in ISO 8601 time format (HH:MM:SS). Example: 14:55:02. This parameter uses the strftime() parameter %T.  
  • Short date. **Format: %F%.** Short date in YYYY-MM-DD format. Example: 2001-08-23. This parameter uses the strftime() parameter %F. **Note:** This parameter will not vary in 24 hours, so use with caution in combination with the rollover time limit feature (time limit should be greater than 1 day; otherwise, you may overwrite the same file contiusly).  
  • Date and time. **Format: %c%.** Date and time representation, locale-dependent. This parameter uses the strftime() parameter %c. **Note:** Using no parameters will yield the same file name every time. Therefore, if a file rollover command is received or scheduled, Recording Service will be stopped (no more data can be stored without overwriting the current, and only, file). Default: rti_recorder_default_%auto:0%.dat (auto-numeric starting at zero, unlimited). | 0..1 |
Table 3.8 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;file-set&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;rollover&gt;</td>
<td>Configuration for rolling over the file after a size or time limit is reached. See Section 3.2.11.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

**3.2.11 Rollover**

Rollover enables *Recording Service* to overwrite the oldest data file created by the current execution of the service when the last file in the set has reached a maximum size, or when a time limit is reached.

**Note:** In this release, rollover is only supported when the `<filename_expression>` tag specifies an auto-numeric filename. (See example below.)

```xml
<recording_service name="RolloverExample">
  <storage>
    <sqlite>
      <storage_format>JSON_SQLITE</storage_format>
      <fileset>
        <workspace_dir>output_dir</workspace_dir>
        <execution_dir_expression></execution_dir_expression>
        <!-- Rollover file when the size hits 50 KB. Files will be numbered 0-9. -->
        <filename_expression>file_rollover_%auto:0-9%.dat</filename_expression>
      </fileset>
    </sqlite>
  </storage>
</recording_service>
```
Table 3.9: Rollover Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;rollover&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;enabled&gt;</td>
<td>Whether Recording Service will roll over files when a limit is reached. Default: False.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;file_size_limit&gt;</td>
<td>The maximum allowed size for a file in a set. Note that setting this to a very low value (e.g., 1 KB) may yield unexpected behavior, because SQLite will take up more than that for even the simplest file. <strong>Note:</strong> The unit refers to the decimal prefix and not the binary prefix of the number, meaning 1 MEGABYTES = 1000 KB (and not 1024 KB). This is usually the standard way to refer to storage size.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| &lt;time_limit&gt;           | The maximum amount of time Recording Service can record to a file in a set. Specified with tags &lt;days&gt;, &lt;hours&gt;, &lt;minutes&gt;, &lt;seconds&gt;. **Attributes:**
  - start_time: The time to do the first rollover. After that, rollover will be done when the time_limit or file_size_limit is reached. | 0..1          |

3.2.12 Plugin

Table 3.10: Plugin Tags in the Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;plugin&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;property&gt;</td>
<td>Name/value pairs of properties to pass to a storage plugin.</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

3.2.13 DomainParticipant

Table 3.11: DomainParticipant Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;domain_participant&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;domain_id&gt;</td>
<td><strong>Required.</strong> DDS domain ID used for recording.</td>
<td>1</td>
</tr>
<tr>
<td>&lt;participant_qos&gt;</td>
<td>QoS used by this DomainParticipant. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.11 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;domain_participant&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <memory_management>              | Configures certain aspects of how Connext DDS allocates internal memory. The configuration is per DomainParticipant and therefore affects all the contained DDS entities. **Example:**  

```
<memory_management>
  <sample_buffer_min_size>
    1024
  </sample_buffer_min_size>
  <sample_buffer_trim_to_size>
    true
  </sample_buffer_trim_to_size>
</memory_management>
```

This tag includes the following tags:

- **sample_buffer_min_size**: For all DataWriters and DataReaders, the way Connext DDS allocates memory for samples is as follows: Connext DDS pre-allots space for samples up to size X in the DataWriter and DataReader queues. If a sample has an actual size greater than X, the memory is allocated dynamically for that sample. The default size is 64KB. This is the maximum amount of pre-allocated memory. Dynamic memory allocation may occur when necessary if samples require a bigger size.

- **sample_buffer_trim_to_size**: If set to true, after allocating dynamic memory for very large samples, that memory will be released when possible. If false, that memory will not be released but kept for future samples if needed. The default is false.

This feature is useful when a data type has a very high maximum size (e.g., megabytes) but most of the samples sent are much smaller than the maximum possible size (e.g., kilobytes). In this case, the memory footprint is reduced dramatically, while still correctly handling the rare cases in which very large samples are published.

| <register_type> | Registers a type name and associates it with a type representation. When you define a type in the configuration file, you have to register the type in order to use it in a <topic>. | 0..* |

### 3.2.14 Session

The `<session>` tag configures the threads that will be used to record data. You also specify the Topics and groups of Topics to record inside the `<session>` tag.
### Table 3.12: Session Tags in Recording Service's Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;session&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;subscriber_qos&gt;</code></td>
<td>Specifies the QoS of DDS subscribers that will be used by the contained <code>&lt;topic&gt;</code> and <code>&lt;topic_group&gt;</code>. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User's Manual.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| `<thread_pool>`         | Defines the number of threads used by this session to process Topics and Topic Groups and sets the mask, priority, and stack size of each thread. **Example:**

```xml
<thread_pool>
  <mask>MASK_DEFAULT</mask>
  <priority>THREAD_PRIORITY_DEFAULT</priority>
  <stack_size>
    THREAD_STACK_SIZE_DEFAULT
  </stack_size>
</thread_pool>
```

**Default values:**
- size: 1
- mask: MASK_DEFAULT
- priority: THREAD_PRIORITY_DEFAULT
- stack_size: THREAD_STACK_SIZE_DEFAULT

| `<topic>`               | Specifies an individual Topic to record. **Attributes:**
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name: The name of the Topic to record. This name is also used when monitoring and administering each Topic.</td>
</tr>
<tr>
<td></td>
<td>participant_ref: A DomainParticipant to use when recording this Topic. If the parent <code>&lt;session&gt;</code> specifies a default_participant_ref, this attribute is optional.</td>
</tr>
</tbody>
</table>

See Section 3.2.16.

| `<topic_group>`         | Specifies a group of Topics to record. **Attributes:**
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name: The name of the Topic group. This name is also used when monitoring and administering each Topic group.</td>
</tr>
<tr>
<td></td>
<td>participant_ref: Specifies a DomainParticipant to use when recording this topic group. If the parent <code>&lt;session&gt;</code> specifies a default_participant_ref, this attribute is optional.</td>
</tr>
</tbody>
</table>

See Section 3.2.15.
### 3.2.15 Topic Group

You can record a group of Topics, using regular expressions to describe which Topics to record.

**Table 3.13: Topic Group Tags in Recording Service’s Configuration File**

<table>
<thead>
<tr>
<th>Tags within &lt;topic_group&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;allow_topic_name_filter&gt;</td>
<td>A regular expression describing which Topics are allowed to be recorded.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;topic_group name=&quot;RecordAll&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;allow_topic_name_filter&gt;**&lt;/allow_topic_name_filter&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/topic_group&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;deny_topic_name_filter&gt;</td>
<td>A regular expression describing which Topics are not allowed to be recorded.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>This tag is applied after the allow_topic_name_filter tag.</td>
<td></td>
</tr>
<tr>
<td>&lt;allow_type_name_filter&gt;</td>
<td>A regular expression describing the names of data types that are allowed to be recorded.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;deny_type_name_filter&gt;</td>
<td>A regular expression describing the names of data types that are not allowed to be recorded.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>This tag is applied after the allow_type_name_filter tag.</td>
<td></td>
</tr>
<tr>
<td>&lt;datareader_qos&gt;</td>
<td>The DataReader’s QoS to use when recording data.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;content_filter&gt;</td>
<td>A ContentFilteredTopic to use when recording data. See ContentFilteredTopics, in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>This allows you to record data samples only if their contents pass a filter that you specify in your configuration. The filter looks like a SQL WHERE clause. Note that XML reserved characters must be converted to their escape values; for example, &gt; and &lt; (greater than and less than signs) must be converted to &lt; and &gt; as shown in the example below. <strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;topic_group name=&quot;RecordAll&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;content_filter&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;!-- Data will only be recorded when x is greater than 100 --&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;expression&gt;x &gt; 100&lt;/expression&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/content_filter&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/topic_group&gt;</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2.16 Topic

The <topic> tag specifies an individual Topic to record.
### Table 3.14: Topic Tags in Recording Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;topic&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;registered_type_name&gt;</code></td>
<td>Required. The name of the data type that will be recorded for this topic.</td>
<td>1</td>
</tr>
</tbody>
</table>
| `<transformation>` | The transformation library to be applied to this Topic’s data when recording. This is a user library that can modify the data after it is received by Connext DDS and before it is stored in the database. Transformations implement APIs identical to Routing Service’s transformations. For more on using transformations, see these sections in the RTI Routing Service User’s Manual:  
  - Data Transformation  
  - Tutorials  
**Attributes:**  
  - plugin_name: The name of the plugin to load, qualified by the plugin library name.  
**Example:**  
```xml  
<dds>  
  <plugin_library name="RecordTransformations">  
    <transformation_plugin name="ModifyTestID">  
      <create_function>ModifyTestID_create</create_function>  
      <dll>modify_test_id_library</dll>  
    </transformation_plugin>  
  </plugin_library>  
  <!-- ... -->  
  <recording_service>  
    <!-- ... -->  
    <topic name="TestTopic">  
      <transformation plugin_name="RecordTransformations::ModifyTestID" />  
    </topic>  
  </recording_service>  
</dds>  
<datareader_qos> | The DataReader QoS to use when recording this data. | 0..1 |

Continued on next page
### Table 3.14 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;topic&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;content_filter&gt;</td>
<td>A ContentFilteredTopic to use when recording data. See ContentFilteredTopics, in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

**Example of how to set a content filter expression:**

```xml
<topic_group name="RecordAll">
  <content_filter>
    <!-- Data will only be recorded when x is greater than 100 -->
    <expression>x &gt; 100</expression>
  </content_filter>
</topic_group>
```

---

**Support for RTI FlatData and Zero Copy Transfer Over Shared Memory**

`Recording Service` supports communication with applications that use RTI FlatData™ and Zero-Copy transfer over shared memory. You can configure `Recording Service` to enable these capabilities for data reception.

To enable `Recording Service` to work with RTI FlatData and Zero-copy transfer over shared memory, you will need to manually define the type in the XML configuration with the proper annotations, and then register this type manually in each `DomainParticipant`. You can use each of these capabilities separately or together.

For further information about these capabilities, see the Sending Large Data section in the `RTI Connext DDS User’s Manual`.

**Example: Configuration to enable both FlatData and zero-copy transfer over shared memory**

```xml
<dds>
  <types>
    <struct name="Point"
      transferMode="shm_ref"
      languageBinding="flat_data"
      extensibility= "final">
      <member name="x" type="long"/>
      <member name="y" type="long"/>
    </struct>
  </types>

  <qos_library name="MyQosLib">
    <qos_profile name="ShmemOnly">
      <participant_qos>
        <discovery>
          <initial_peers>
            <element>shm://</element>
          </initial_peers>
        </discovery>
      </participant_qos>
    </qos_profile>
  </qos_library>
</dds>
```

(continues on next page)
3.2.17 Enabling 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 for the set of platforms that support Distributed Logger.

When you enable Distributed Logger, the Service will publish its log messages to Connext DDS. Then you can use RTI Admin Console to visualize the log message data. Since the data is provided in a topic, you can also use rtiddsspy or even write your own visualization tool.

To enable Distributed Logger, use the tag <distributed_logger> within <administration>. For example:

```
<recording_service name="RecordAll">
  <administration>
    ...
    <distributed_logger>
      <enabled>true</enabled>
    </distributed_logger>
    ...
  </administration>
</recording_service>
```

For more details, see Enabling Distributed Logger in RTI Services, in the RTI Connext DDS Core Libraries User’s Manual.
3.2.18 Recording Service Builtin Configuration Details

The Recording Service builtin configuration specifies:

- Recording all non-RTI Topics
- In domain 0
- Into a SQLite file named rti_recorder_default.dat
- In the efficient XCDR format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../schema/rti_recording_service.xsd">
    <!-- Qos Library -->
    <qos_library name="QosLib">
        <qos_profile name="ReliableQos">
            <datareader_qos>
                <reliability>
                    <kind>RELIABLE_RELIABILITY_QOS</kind>
                </reliability>
                <history>
                    <kind>KEEP_ALL_HISTORY_QOS</kind>
                </history>
            </datareader_qos>
        </qos_profile>
    </qos_library>

    <recording_service name="RecorderService">
        <!-- Top-level storage settings -->
        <storage>
            <sqlite>
                <file>rti_recorder_default</file>
                <file_suffix>dat</file_suffix>
                <storage_format>XCDR_AUTO</storage_format>
            </sqlite>
        </storage>

        <!-- Top-level domain settings -->
        <domain_participant name="Participant0">
            <domain_id>0</domain_id>
        </domain_participant>

        <session name="DefaultSession">
            <topic_group name="RecordAll" participant_ref="Participant0">
                <allow_topic_name_filter>*</allow_topic_name_filter>
                <deny_topic_name_filter>rti/</deny_topic_name_filter>
                <datareader_qos base_name="QosLib::ReliableQos" />
            </topic_group>
        </session>
    </recording_service>
</dds>
```

(continues on next page)
3.3 Remote Administration

A control client (such as RTI Admin Console) can use this interface to remotely control Recording Service.

**Note:** Recording Service remote administration is based on the RTI Remote Administration Platform described in Section 6.2. Please refer to that manual for a detailed discussion on the workings of remote administration in Recording Service.

Below you will find an API reference for all the supported operations.

### 3.3.1 Enabling Remote Administration

By default, remote administration is disabled in Recording Service.

To enable remote administration you can use the `<administration>` XML tag (see Section 3.2.6) or the `-remoteAdministrationDomainId` command-line parameter (see Section 3.1.3). Both of these methods enable remote administration and set the domain ID for remote communication.

### 3.3.2 Available Service Resources

Table 3.15 lists the public resources specific to Recording Service. Each resource identifier is expressed as a hierarchical sequence of identifiers, including parent and target resources. (See Section 6.1.1 for details.)

In the table below, the elements (rs), and (st) refer to the name of an entity of the corresponding class as specified in the configuration in the name attribute. For example, in the following configuration:

```xml
<recording_service name="MyRecorder">...</recording_service>
```

The resource identifier is:

```
/recording_services/MyRecorder
```

In the table below, the resource identifier is written as `/recording_services/(rs)`, where (rs) is the service name. (st) is the storage name, and so on. This nomenclature is used in the table to give you an idea of the structure of the resource identifiers. For actual (example) resource identifier names, see the example section that follows.
Table 3.15: Resources and Their Identifiers in Recording Service

<table>
<thead>
<tr>
<th>Resource</th>
<th>Resource Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recording Service</td>
<td>/recording_services/(rs)</td>
</tr>
<tr>
<td>Storage</td>
<td>/recording_services/(rs)/storage/(st)</td>
</tr>
</tbody>
</table>

**Example**

This example shows you how to address a resource of each possible resource class in Recording Service.

**Recording Service**

Entity with name “MyRecorder”:

```xml
text
<recording_service name="MyRecorder">...</recording_service>
```

Resource identifier:

```
/recording_services/MyRecorder
```

**Storage**

Entity with name “sqlite” (implicit name of the builtin storage):

```xml
text
<recording_service name="MyRecorder">
  <sqlite>...</sqlite>
</recording_service>
```

Resource identifier:

```
/recording_services/MyRecorder/storage/sqlite
```
3.3.3 Remote API Overview

Table 3.16: Remote Interface Overview

<table>
<thead>
<tr>
<th>Resource</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recording Service</td>
<td>DELETE /recording_services/(rs)</td>
<td>Shuts down a running Recording Service instance.</td>
</tr>
<tr>
<td></td>
<td>UPDATE /recording_services/(rs)/state</td>
<td>Sets a Recording Service state.</td>
</tr>
<tr>
<td>builtin SQLite Storage</td>
<td>UPDATE /recording_services/(rs)/storage/sqlite:rollover</td>
<td>Continues the current recording in a new file (also known as a “shard” or file segment). Note: Only valid for builtin SQLite plugin.</td>
</tr>
<tr>
<td></td>
<td>UPDATE /recording_services/(rs)/storage/sqlite:tag_timestamp</td>
<td>Associates a symbolic name with the current time (with an optional offset) in the recording. Note: Only valid for builtin SQLite plugin.</td>
</tr>
</tbody>
</table>

3.3.4 Recording Service

DELETE /recording_services/(rs)
Operation shutdown

This operation will cause Recording Service to shutdown.

UPDATE /recording_services/(rs)/state
Operation set_state

Sets the state of a Recording Service object. The action is parametrized on octet_body, which could have the following values:

See Set Resource State (Section 6.2.3).

Valid requested states:

- STARTED
- STOPPED
- PAUSED
- RUNNING

Example

To pause an instance of Recording Service with the name “MyRecorder”: 
3.3.5 Storage

**UPDATE /recording_services/(rs)/storage/sqlite:rollover**

**Operation rollover**

If the storage plugin being used is the builtin SQLite plugin, this operation will cause the recording to switch to a new file segment (also known as a “shard”) in the files.

<table>
<thead>
<tr>
<th>Request Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>UPDATE</td>
</tr>
<tr>
<td>resource_identifier</td>
<td>/recording_services/MyRecorder/storage/sqlite:rollover</td>
</tr>
<tr>
<td>octet_body</td>
<td>to_cdr_buffer(RTI::Service::EntityStateKind::PAUSED)</td>
</tr>
</tbody>
</table>

**Request body**

- Empty.

**UPDATE /recording_services/(rs)/storage/sqlite:tag_timestamp**

**Operation tag_timestamp**

If the storage plugin being used is the builtin SQLite plugin, this operation will introduce a new entry in the symbolic timestamps table, using the tag name, textual description, and offset time specified in the arguments to the operation (CDR-serialized in the octet-body with the RTI::RecordingService::DataTagParams data type).

- Example

To tag a moment with the name “/example/test1/tag1”, representing a moment that is 123 milliseconds in the past:
### 3.4 Monitoring

This section provides documentation on Recording Service remote monitoring.

**Note:** Recording Service monitoring is based on the Monitoring Distribution Platform described in Section 6.3. We recommend that you read Section 6.3 before using Recording Service monitoring.

#### 3.4.1 Overview

**Enabling Service Monitoring**

By default, monitoring is disabled in Recording Service. To enable monitoring you can use the `<monitoring>` tag (see Section 3.2.5) or the `-remoteMonitoringDomainId` command-line parameter, which enables remote monitoring and sets the domain ID for data publication (see Section 3.1.3).

**Monitoring Types**

The available Keyed Resource classes and their types that can be present in the distribution monitoring topics are listed in Table 3.17. The complete type relationship is shown in Figure 3.3.

<table>
<thead>
<tr>
<th>Keyed Resource Class</th>
<th>Config</th>
<th>Event</th>
<th>Periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>ServiceConfig</td>
<td>ServiceEvent</td>
<td>ServicePeriodic</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.17 – continued from previous page

<table>
<thead>
<tr>
<th>Keyed Resource Class</th>
<th>Config</th>
<th>Event</th>
<th>Periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Session</strong></td>
<td>SessionConfig</td>
<td>SessionEvent</td>
<td>SessionPeriodic</td>
</tr>
<tr>
<td><strong>TopicGroup</strong></td>
<td>TopicGroupConfig</td>
<td>TopicGroupEvent</td>
<td>TopicGroupPeriodic</td>
</tr>
<tr>
<td><strong>Topic</strong></td>
<td>TopicConfig</td>
<td>TopicEvent</td>
<td>TopicPeriodic</td>
</tr>
</tbody>
</table>

Figure 3.3: Keyed Resource Types for Recording Service monitoring

All the type definitions for Recording Service monitoring information are in \[\text{NDDSHOME}/resource/idl/ServiceCommon.idl\] and \[\text{NDDSHOME}/resource/idl/RecordingServiceMonitoring.idl\].

Recording Service creates a DataWriter for each distribution Topic. All DataWriters are created from a single Publisher, which is created from a dedicated DomainParticipant. See Section 3.2.5 for details on configuring the QoS for these entities.

### 3.4.2 Monitoring Metrics Reference

This section provides a reference to all the monitoring metrics Recording Service distributes, organized by service resource class.

**Service**

Listing 3.1: Recording Service Types

```cpp
@mutable @nested
struct SqliteDatabaseConfig {
    Service::FilePath db_directory;
}
```

(continues on next page)
@mutable @nested
struct SqliteDatabaseEvent {
    @optional Service::FilePath current_db_directory;
    @optional Service::FilePath current_file;
    @optional int32 rollover_count;
};

@mutable @nested
struct SqliteDatabasePeriodic {
    @optional Service::FilePath current_file;
    @optional uint64 current_file_size;
    int32 current_timestamp_sec;
    uint32 current_timestamp_nanosec;
};

@mutable @nested
struct ParticipantInfo {
    Service::BoundedString name;
};

@mutable @nested
struct ServiceConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString application_name;
    Service:: Monitoring::ResourceGuid application_guid;
    @optional Service::Monitoring::HostConfig host;
    @optional Service::Monitoring::ProcessConfig process;
    @optional SqliteDatabaseConfig builtin_sqlite;
    @optional sequence<ParticipantInfo> participants;
};

@mutable @nested
struct ServiceEvent : Service::Monitoring::EntityEvent {
    //to avoid unused variable warnings
    @optional SqliteDatabaseEvent builtin_sqlite;
};

@mutable @nested
struct ServicePeriodic {
    @optional Service::Monitoring::HostPeriodic host;
    @optional Service::Monitoring::ProcessPeriodic process;
    @optional SqliteDatabasePeriodic builtin_sqlite;
};
Table 3.18: ServiceConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>application_name</td>
<td>Name of the Recording Service instance. The application name is provided through:</td>
</tr>
<tr>
<td></td>
<td>• appName command-line option when run as executable.</td>
</tr>
<tr>
<td></td>
<td>• ServiceProperty::application_name field when run as a library.</td>
</tr>
<tr>
<td>application_guid</td>
<td>GUID of the Recording Service instance. Unique across all service instances.</td>
</tr>
<tr>
<td>host</td>
<td>See Table 6.9.</td>
</tr>
<tr>
<td>process</td>
<td>See Table 6.11.</td>
</tr>
<tr>
<td>builtin_sqlite</td>
<td>See Table 3.19</td>
</tr>
<tr>
<td>participants</td>
<td>Sequence of ParticipantInfo objects, one for each DomainParticipant inside the Recording Service. See Table 3.20.</td>
</tr>
</tbody>
</table>

Table 3.19: SqliteDatabaseConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_directory</td>
<td>Path to the base directory where the database files will live. This is the prefix directory, and still the execution directory expression below will be appended to it to determine the final directory.</td>
</tr>
<tr>
<td>execution_directory_expression</td>
<td>The expression used to generate the directory where the database files live. Note: this value is not set when running Replay Service. See Section 3.2.10</td>
</tr>
<tr>
<td>user_data_file_expression</td>
<td>The expression used to generate the names of the database files. Note: this value is not set when running Replay Service. See Section 3.2.10</td>
</tr>
</tbody>
</table>

Table 3.20: ParticipantInfo

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the DomainParticipant instance, as specified in the name attribute of the corresponding configuration tag.</td>
</tr>
</tbody>
</table>

Table 3.21: ServiceEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
<tr>
<td>builtin_sqlite</td>
<td>See Table 3.22</td>
</tr>
</tbody>
</table>

3.4. Monitoring

37
Table 3.22: SqliteDatabaseEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_directory</td>
<td>Path to the current directory where files are being stored to. This is the actual final directory once the base directory and any execution directory expressions have been applied.</td>
</tr>
<tr>
<td>current_file</td>
<td>Path to the current file where data is being stored to. Note: this value is not set when running Replay Service.</td>
</tr>
<tr>
<td>rollover_count</td>
<td>The total number of file rollover events that have happened to this moment. Note: this value is not set when running Replay Service.</td>
</tr>
</tbody>
</table>

Table 3.23: ServicePeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>See Table 6.10.</td>
</tr>
<tr>
<td>process</td>
<td>See Table 6.12.</td>
</tr>
<tr>
<td>builtin_sqlite</td>
<td>See Table 3.24</td>
</tr>
</tbody>
</table>

Table 3.24: SqliteDatabasePeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_file_size</td>
<td>The size in bytes of the current file where data is being stored. Note: this value is not set when running Replay Service.</td>
</tr>
<tr>
<td>current_timestamp_sec</td>
<td>This time represents the seconds Recording Service has been running.</td>
</tr>
<tr>
<td>current_timestamp_nanosec</td>
<td>This time represents the nanoseconds Recording Service has been running.</td>
</tr>
</tbody>
</table>

Session

Listing 3.2: Session Types

```cpp
@mutable @nested
struct SessionConfig : Service::Monitoring::EntityConfig {
  Service::BoundedString default_participant_name;
};

@mutable @nested
struct SessionEvent : Service::Monitoring::EntityEvent {
  //to avoid unused variable warnings
  int32 _dummy;
};

@mutable @nested
struct SessionPeriodic {
  @optional Service::Monitoring::NetworkPerformance network_performance;
};
```

3.4. Monitoring
### Table 3.25: SessionConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>default_participant_name</td>
<td>The name of the default participant configuration.</td>
</tr>
</tbody>
</table>

### Table 3.26: SessionEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
</tbody>
</table>

### Table 3.27: SessionPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_performance</td>
<td>Provides network performance metric as an aggregation of the same metric across the contained Topic and TopicGroups. See Section 6.3.4.</td>
</tr>
</tbody>
</table>

### TopicGroup

### Listing 3.3: TopicGroup Types

```cpp
@mutable @nested
g struct TopicGroupConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString participant_name;
};

@mutable @nested
g struct TopicGroupEvent : Service::Monitoring::EntityEvent {
    // to avoid unused variable warnings
    int32 _dummy;
};

@mutable @nested
g struct TopicGroupPeriodic {
    @optional Service::Monitoring::NetworkPerformance network_performance;
    int64 topic_count;
};
```

### Table 3.28: TopicGroupConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>participant_name</td>
<td>Name of the DomainParticipant from which the Topic is created.</td>
</tr>
</tbody>
</table>
Table 3.29: TopicGroupEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
</tbody>
</table>

Table 3.30: TopicGroupPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_performance</td>
<td>Provides network performance metric as an aggregation of the same metric across the contained Topic. See Section 6.3.4.</td>
</tr>
<tr>
<td>topic_count</td>
<td>Current number of Topic created from this TopicGroup.</td>
</tr>
</tbody>
</table>

Topic

Listing 3.4: Topic Types

```cpp
@mutable @nested
struct TopicConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString topic_name;
    Service::BoundedString registered_type_name;
    Service::BoundedString participant_name;
    Service::Monitoring::ResourceGuid topic_group;
};

@mutable @nested
struct TopicEvent : Service::Monitoring::EntityEvent {
    // to avoid unused variable warnings
    int32 _dummy;
};

@mutable @nested
struct TopicPeriodic {
    @optional Service::Monitoring::NetworkPerformance network_performance;
    @optional Service::Monitoring::CountStatus matched_status;
};
```

Table 3.31: TopicConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic_name</td>
<td>Topic name as specified in the configuration.</td>
</tr>
<tr>
<td>registered_type_name</td>
<td>Topic registered type name as specified in the configuration.</td>
</tr>
<tr>
<td>participant_name</td>
<td>Name of the DomainParticipant from which the Topic is created.</td>
</tr>
<tr>
<td>topic_group</td>
<td>GUID of the TopicGroup from which this Topic was created. This field is set to zero for standalone Topic.</td>
</tr>
</tbody>
</table>

3.4. Monitoring
Table 3.32: TopicEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
</tbody>
</table>

Table 3.33: TopicPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_performance</td>
<td>Provides network performance metric as an aggregation of the same metric across the contained Topic. See Section 6.3.4.</td>
</tr>
<tr>
<td>matched_status</td>
<td>Provides information about the matched endpoints associated with this Topic.</td>
</tr>
</tbody>
</table>

3.5 Tutorials

3.5.1 Getting Started with Recording Service and Shapes Demo

In this tutorial, you will edit a Recording Service configuration file to record only specific topics. Then you will start the Shapes Demo application and publish multiple topics. You will then inspect the database to see that only the specified topics were recorded.

Edit the Configuration

The first time you run any RTI tools, an rti_workspace directory is created in your home directory. (See Section 1.3 for the path to your workspace).

Inside the workspace, you will find a directory called user_config/recording_service. Open the USER_RECORDING_SERVICE.xml file in an editor. Notice that there are two <recording_service> tags in this file, one named UserRecorderService and one named UserRecorderServiceJson. The UserRecorderServiceJson configuration is shown below.

```xml
<recording_service name="UserRecorderServiceJson">
  <!-- Top-level storage settings -->
  <storage>
    <sqlite>
      <storage_format>JSON_SQLITE</storage_format>
    </sqlite>
    <fileset>
      <workspace_dir></workspace_dir>
      <execution_dir_expression>json_recording</execution_dir_expression>
      <filename_expression>rti_recorder_default_json.dat</filename_expression>
    </fileset>
  </storage>
  <!-- Top-level domain settings -->
  <domain_participant name="Participant0">
    <domain_id>0</domain_id>
  </domain_participant>
</recording_service>
```

(continues on next page)
This configuration is recording all topics in domain 0 into a database file named rti_recorder_default_json.dat. It is recording in deserialized (JSON_SQLITE) mode. In the configuration, change the value in `<allow_topic_name_filter>` from `*` to `Square`. Now it will record only the Square topic.

```xml
<recording_service name="UserRecorderServiceJson">
  <!-- Top-level storage settings -->
  <storage>
    <sqlite>
      <storage_format>JSON_SQLITE</storage_format>
      <fileset>
        <workspace_dir/></workspace_dir>
        <execution_dir_expression>json_recording</execution_dir_expression>
        <filename_expression>rti_recorder_default_json.dat</filename_expression>
      </fileset>
    </sqlite>
  </storage>
  <!-- Top-level domain settings -->
  <domain_participant name="Participant0">
    <domain_id>0</domain_id>
  </domain_participant>
  <session name="DefaultSession">
    <topic_group name="RecordAll" participant_ref="Participant0">
      <allow_topic_name_filter>*</allow_topic_name_filter>
      <deny_topic_name_filter>rti/*</deny_topic_name_filter>
    </topic_group>
  </session>
</recording_service>
```

Start Shapes Demo

Start Shapes Demo from the Launcher tool.

In Shapes Demo, start two publishers: One Square publisher and one Circle publisher.

Start Recording Service

Start Recording Service with the UserRecorderServiceJson configuration and with verbosity level 3 by entering the following in a command shell:

```
```
cd <RTI_WORKSPACE>/user_config/recording_service  
<NDDSHOME>/bin/rtirecordingservice -cfgName UserRecorderServiceJson -verbosity 3

See Section 1.3 for the path to your workspace and NDDSHOME.

You should see output indicating that the Square topic is being recorded:

[PARTICIPANT id=/recording_services/UserRecorderServiceJson/domain_participants/  
→Participant0|STREAM_DISCOVERED name=Square|TOPIC id=/recording_services/  
→UserRecorderServiceJson/sessions/DefaultSession/topics/RecordAll@Square|CREATE]

[PARTICIPANT id=/recording_services/UserRecorderServiceJson/domain_participants/  
→Participant0|STREAM_DISCOVERED name=Square|TOPIC id=/recording_services/  
→UserRecorderServiceJson/sessions/DefaultSession/topics/RecordAll@Square|ENABLE]

[TOPIC id=/recording_services/UserRecorderServiceJson/sessions/DefaultSession/topics/  
→RecordAll@Square|START]

[TOPIC id=/recording_services/UserRecorderServiceJson/sessions/DefaultSession/topics/  
→RecordAll@Square|RUN]

View the Data in Sqlite3

First, download the sqlite3 command-line browser from the SQLite download page. (If you are using a Linux or Mac OS system, you may already have this installed.)

Then open a command prompt and run the application:

sqlite3 rti_recorder_default_json.dat

Type the command .tables and you should see recorded data for topic Square:

sqlite> .tables
    Square@0

Then you can view your recorded Square data by typing:

sqlite> select rti_json_sample from "Square@0";

{"color":"ORANGE","x":120,"y":195,"shapesize":30,"fillKind":"SOLID_FILL","angle":0}
{"color":"ORANGE","x":120,"y":197,"shapesize":30,"fillKind":"SOLID_FILL","angle":0}
{"color":"ORANGE","x":120,"y":199,"shapesize":30,"fillKind":"SOLID_FILL","angle":0}
{"color":"ORANGE","x":120,"y":201,"shapesize":30,"fillKind":"SOLID_FILL","angle":0}

3.5.2 Using Recording Service and Admin Console

You can use RTI Admin Console to monitor, pause and resume Recording Service.

Configuration

To use Recording Service with Admin Console, make sure that administration is enabled in the configuration. For example:
<recording_service name="AdminExample">
  <administration>
    <domain_id>0</domain_id>
  </administration>
  ...
</recording_service>

Note that enabling administration will also enable monitoring in the same domain by default. Admin Console cannot control the Recording Service instance unless monitoring is enabled.

**Start Recording Service**

*Admin Console* cannot start a new instance of *Recording Service*. It can only monitor and administer a instance of *Recording Service* that is already running.

To begin, start *Recording Service* with administration enabled:

```
<NDDSHOME>/bin/rtirecordingservice -cfgName AdminExample
```

**Start Shapes Demo**

Start *Shapes Demo* and publish Squares and Circles as described in Section 3.5.1.

**Viewing with Admin Console**

Use the *Launcher* tool to start *Admin Console* (from the Tools tab).

You will see the Recording Service appear in *Admin Console’s* Physical View and Processes View. Click on the Recording Service in either view and a Recording Service tab will appear.

The first tab, **Recording Entities**, shows your Recording Service, the session(s) it is running, the topics and topic groups it is configured to record, and which topics are being recorded.

Note that topics that are being recorded as part of a topic_group will appear side-by-side with topics that were configured individually. You can tell they are part of a topic_group because the name of the topic_group will appear along with the topic. For example, *RecordAll@Square*.

The second tab, **Recording Configuration**, displays the configuration that was used to configure the running Recording Service instance.

The third tab, **DDS Entities**, displays the DDS entities that have been created by the Recording Service. If you click on individual DataReaders, you can view their QoS in the DDS QoS view.

The fourth tab, **Recording Information**, shows details about the recording, such as:

- The *Connext DDS* version number
- The name of the database file it is recording (if you are using the builtin SQLite storage)
- The current size of the database file you are recording (if you are using the builtin SQLite storage)

The final tab, **Resource Charts**, allows you to monitor *Recording Service’s* CPU and memory usage.
3.5. Tutorials
3.5. Tutorials
3.5. Tutorials
Administering with Admin Console

*Recording Service* allows the following commands:

- **Pause:** This pauses all the topics in a running service.
- **Resume:** This restarts the sessions and topics in a *Recording Service* application.
- **Shut down:** This shuts down the *Recording Service* application. To restart the application, you must re-run from *Launcher* or the command line.

3.5.3 Using Recording Service as a Library

It is possible to use *Recording Service* as a library in your application. All the necessary tools are included in the library `rtirecordingservice` (*librtirecordingservice.so* on Linux, *librtirecordingservice.dylib* on Mac OS X and *rtirecordingservice.dll* on Windows). The library can be used in any of the modes provided by *Recording Service*: recording data, replaying data or converting data. For more information, see: *Recording Service C++ API*.

A simple C++ example of how to link the Recording Service library into your application is available here: RTI Community Recording Service examples: Service as a Library

**Include files**

When using *Recording Service* as a library, you should include the following header in your application code:

```cpp
#include <rti/recording/RecordingService.hpp>
```
Using the RecordingService class

The main class used to instantiate Recording Service as a library is called RecordingService. To instantiate it, you need to define the properties to be used by the service for its configuration. The class used to define these properties is called ServiceProperty. Among the parameters that can be set to control the service’s execution, you can find:

- Application role: record, replay or convert.
- Configuration control: configuration file as well as configuration name (service name).
- Remote administration control: flag to enable or disable it as well as the domain ID to use.
- Monitoring control: flag to enable or disable it as well as the domain ID to use.
- DDS Domain ID control: the domain ID base parameter allows the user to apply an offset to the domain IDs specified in the XML configuration to be loaded.
- XML control: whether to load the default Connext XML files or not or whether to enforce XSD validation of the supplied configuration file.

The following code snippet shows how to launch a Recording Service instance in record mode. This instance will use an XML configuration called MyRecorderConfig that can be found in file my_recording_service.xml. Administration and monitoring will be enabled and attached to domain ID 5. By using the domain ID base as well, all the domain IDs specified in the aforementioned XML configuration will be offset by 6.

```cpp
using namespace rti::recording;

// ...

ServiceProperty service_property;
service_property.application_role(ApplicationRoleKind::RECORD_APPLICATION).
    enable_administration(true).
    administration_domain_id(5).
    enable_monitoring(true).
    monitoring_domain_id(5).
    domain_id_base(6).
    cfg_file("my_recording_service.xml").
    service_name("MyRecorderConfig");
RecordingService recorder(service_property);
recorder.start();

// ...

// When it's time to stop the Recording Service instance, call the method
// below. The "stop()" method will also be called automatically by the
// RecordingService class destructor, so it will be called when the
// instance goes out of scope.
recorder.stop();
```

3.5.4 Plugging in Custom Storage
Note: The APIs for custom storage are not available in the Basic bundle.

To configure a custom storage library, you must add the `<plugin_library>` tag inside the `<dds>` tag. This allows you to define one or more storage libraries that can be used to plug in custom storage.

In addition, you must associate the storage library with your Recording Service instance by specifying the plugin and its name in the `<storage>` tag.

You can pass custom properties to your plugin inside the `<plugin>` tag.

There are full examples written in C and C++ about plugging in custom storage in Recording Service, in: RTI Community Recording Service examples: C storage plugin and RTI Community Recording Service examples: C++ storage plugin.

Custom Storage API Overview

To store data, you must implement the following APIs:

- RecordingServiceStorageWriter create function: A creation function for the StorageWriter structure or class. This allocates a StorageWriter object, which is used to allocate StorageStreamWriters.

- StorageWriter:
  - A function for creating StorageStreamWriters for user-data topics when Recording Service notifies the plugin about a new stream. The user-data streams represent samples as Dynamic Data objects.
  - Three functions for creating StorageStreamWriters for the built-in discovery topics: DCPSParticipant, DCPSPublication and DCPSSubcription. These topics are represented by their specific types (e.g., DDS_ParticipantBuiltinTopicData type for DCPSParticipant). These functions are not required, thus when one of them is not implemented, no samples will be stored for that built-in topic.
  - A function for deleting user-data StorageStreamWriters. Recording Service expects this function to be able to work with streams writers created for user-data samples only.
  - Three functions to delete the StorageStreamWriters representing the DDS built-in topics. These functions are not required, but if the creation function was defined for a topic, the deletion function must also be defined.
  - A function for deleting the StorageWriter instance.

- StorageStreamWriter:
  - A function for storing data associated with a stream. For the stored data to be compatible with Replay Service, the reception timestamp of every sample should be stored with the data. It is also recommended that the valid data flag is recorded. For Replay Service to be able to replay instance states, it’s also necessary for Recording Service to store the instance state and instance handle fields.
There are three specific classes for the three different builtin discovery topics: `StorageParticipantWriter`, `StoragePublicationWriter` and `StorageSubscriptionWriter`. Each of them includes a `store()` function that is strongly typed to the topic’s type.

The C++ APIs provide a mechanism to have strongly typed `StorageStreamWriter` classes. There are four specific builtin `StorageStreamWriter` definitions to work with the different types of streams:

- A definition based on `dds::core::xtypes::DynamicData`, which should be used to store samples for user-data topics.
- A definition based on `dds::topic::ParticipantBuiltinTopicData`, which can be used to store samples of the builtin DDS discovery stream `DCPSParticipant`.
- A definition based on `dds::topic::PublicationBuiltinTopicData`, which can be used to store samples of the builtin DDS discovery stream `DCPSPublication`.
- A definition based on `dds::topic::SubscriptionBuiltinTopicData`, which can be used to store samples of the builtin DDS discovery stream `DCPSSubscription`.

More detailed API documentation is located here:

- Recording Service C API documentation
- Recording Service C++ API documentation

### 3.5.5 Accessing JSON samples through SQL

When using the builtin SQLite JSON storage format, data samples are stored in the column called `rti_json_sample` using SQLite’s JSON extensions. The sample can thus be accessed using these extensions, namely, the `json_extract()` function.

As an example, suppose we have the following IDL type:

```c
struct BasicStruct {
  double member1;
  string member2;
};
```

We could access the sample's data with a SQL query like this:

```
SELECT json_extract(rti_json_sample, '$.member2') FROM [MyTableName]
WHERE json_extract(rti_json_sample, '$.member1') > 2.0
```

### 3.5.6 Controlling Recording Service Remotely from an Application

Apart from the ability to use Admin Console to control a Recording Service instance, it is possible to control it using an application that issues command requests programmatically, using the Remote Administration Platform.

There is a C++ example in the RTI Community that provides an application that can produce command requests for Recording Service (or in general, for any RTI service that uses the common Remote Administration Platform): RTI Community Examples: C++ Service Administration.
3.5.7 Listing the Timestamp Tags in a Recording

If you are using the built-in SQLite plugin in Recording Service, and remote administration is enabled, then the remote administration exposes a command that allows you to tag timestamps with a symbolic name and description. These timestamp tags allow you to associate points in time in the recording with external events. The tags can be used later with Replay Service or Converter in place of timestamps to select what data to replay or convert.

The use case for timestamp tags is that they allow you to record, in a human-readable way, the time at which interesting events occur in your business processes along with the database that Recording Service is recording into. Later you can use these markers to replay data from (or up to) that point, instead of expressing start and end times in a numeric way.

For example, suppose you use Recording Service to record data while you are doing some business process. Then some event happens during the process that you want to somehow mark in the database. (Maybe this event marks the time at which some device starts behaving strangely during a troubleshooting session). If you create a custom GUI application that an operator can use, or any kind of application, you can use the timestamp tagging command described in this section. Then when that application sends the command, Recording Service will store in the database a record in which you can give the event a name and description. Later you can replay the recorded data starting (or ending) at that named event.

Note that you can use a time offset when submitting the timestamp tagging command. This allows you to create tags that refer to a time in the past or the future, relative to the time when you sent the tagging command.

Once you have a recorded database, you can list the timestamp tags that are in the recording. Use the command rtirecordingservice_list_tags and point it to the directory that contains your recorded database with the -d argument.

For example:

```
<NDDSHOME>/bin/rtirecordingservice_list_tags -d /database/directory/
```

This command will analyze the recording in /database/directory/ and list the details of any timestamp tags in the recording, including the tag names, descriptions, and associated timestamps. For example:

```
tag_name                timestamp_ms    tag_description
----------------------------  --------------  ------------------------
/my_example/my_events/tag1  154648463309  first tag description
/my_example/my_events/tag2  1546484703360  a second tag description
```

For information on using timestamp tags with Replay Service, see Section 4.5.4. They can also be used with Converter, see Section 5.3.1.

3.6 Troubleshooting

3.6.1 Verbosity

Increase the verbosity from the command line to view information about which Topics have been discovered. For example, running with -verbosity 4 shows that Recording Service has discovered
a Square Topic.

At verbosity level 4, it is also possible to see incompatible QoS.

PRESPsService_isRemoteWriterLocalReaderCompatible: incompatible ownership: writer 1 ⊣ reader 0

At verbosity level 4, it is also possible to see incompatible QoS.
Chapter 4

Replay Service

4.1 Usage

This section explains how to run Replay Service from a command line. In particular, it describes:

- How to Start Replay Service (Section 4.1.1).
- How to Stop Replay Service (Section 4.1.2).
- Replay Service command-line parameters (Section 4.1.3).

4.1.1 Starting Replay Service

Replay Service runs as a separate application. The script to run the executable is in <NDDSHOME>/bin. (See Section 1.3 for the path to NDDSHOME.)

```
rtireplayservice [options]
```

To start Replay Service with a default configuration, enter:

```
$NDDSHOME/bin/rtireplayservice
```

This command will run Replay Service indefinitely until you stop it.

**Note:** Replay Service is pre-loaded with a built-in configuration that has default settings.

4.1.2 Stopping Replay Service

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

4.1.3 Replay Service Command-Line Parameters

The following table describes all the command-line parameters available in Replay Service. To list the available parameters, run rtireplayservice -help.
All command-line parameters are optional; if specified, they override the values of any corresponding settings in the loaded XML configuration. See Section 4.2.3 for the XML elements that can be overridden with command-line parameters.

Table 4.1: Replay Service Command-Line Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-appName &lt;string&gt;</td>
<td>Application name used to identify this execution for remote administration, and to name the Connext DDS participant.</td>
</tr>
<tr>
<td>-cfgFile &lt;string&gt;</td>
<td>Semicolon-separated list of configuration file paths. Default: Unspecified</td>
</tr>
<tr>
<td>-cfgName</td>
<td>Configuration name Used to find a &lt;replay_service&gt; matching tag in the configuration file.</td>
</tr>
<tr>
<td>-domainIdBase &lt;int&gt;</td>
<td>This value is added to the domain IDs in the &lt;domain_participant&gt; tag in the configuration file. For example, if you set -domainIdBase to 50 and use domain IDs 0 and 1 in the configuration file, Replay Service will read domains 0 and 1 from the database, but will replay that data into domains 50 and 51. Default: 0</td>
</tr>
<tr>
<td>-D&lt;name&gt;=&lt;value&gt;</td>
<td>Defines a variable that can be used as an alternate replacement for XML environment variables, specified in the form $(VAR_NAME). Note that definitions in the environment take precedence over these definitions.</td>
</tr>
<tr>
<td>-help</td>
<td>Shows this help.</td>
</tr>
<tr>
<td>-heapSnapshotDir</td>
<td>Output directory where the heap monitoring snapshots are dumped. The filename format is: RTI_heap_&lt;appName&gt;<em>&lt;processId&gt;</em>&lt;index&gt;.log</td>
</tr>
<tr>
<td>-heapSnapshotPeriod &lt;sec&gt;</td>
<td>Period at which heap monitoring snapshots are dumped. Enables heap monitoring if &gt; 0. Default: 0 (disabled)</td>
</tr>
<tr>
<td>-logFormat &lt;format&gt;</td>
<td>A mask to configure the format of the log messages for both Replay Service and Connext DDS.</td>
</tr>
<tr>
<td></td>
<td>• DEFAULT - Print message, method name, and activity context</td>
</tr>
<tr>
<td></td>
<td>• TIMESTAMPED - Print message, method name, activity context, and timestamp</td>
</tr>
<tr>
<td></td>
<td>• MINIMAL - Print only message number and method name</td>
</tr>
<tr>
<td></td>
<td>• MAXIMAL - Print all available fields</td>
</tr>
<tr>
<td>Default: DEFAULT</td>
<td></td>
</tr>
<tr>
<td>-maxObjectsPerThread &lt;int&gt;</td>
<td>Maximum number of Thread-specific objects that can be created. Default: 1024.</td>
</tr>
<tr>
<td>-remoteAdministrationDomainId &lt;int&gt;</td>
<td>Enables remote administration and sets the domain ID for communication. Default: Remote administration is not enabled.</td>
</tr>
<tr>
<td>-remoteMonitoringDomainId &lt;int&gt;</td>
<td>Enables remote monitoring and sets the domain ID for status publication. Default: Remote monitoring is not enabled.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-verbosity [0-6]</td>
<td><strong>RTI Service verbosity</strong></td>
</tr>
<tr>
<td></td>
<td>• 0 - silent</td>
</tr>
<tr>
<td></td>
<td>• 1 - exceptions <em>(Connext DDS and Replay Service)</em></td>
</tr>
<tr>
<td></td>
<td>• 2 - warnings <em>(Replay Service)</em></td>
</tr>
<tr>
<td></td>
<td>• 3 - information <em>(Replay Service)</em></td>
</tr>
<tr>
<td></td>
<td>• 4 - warnings <em>(Connext DDS and Replay Service)</em></td>
</tr>
<tr>
<td></td>
<td>• 5 - tracing <em>(Replay Service)</em></td>
</tr>
<tr>
<td></td>
<td>• 6 - tracing <em>(Connext DDS and Replay Service)</em></td>
</tr>
<tr>
<td>Default: 1</td>
<td>(exceptions)</td>
</tr>
<tr>
<td>-version</td>
<td>Prints the program version and exits.</td>
</tr>
</tbody>
</table>

### 4.1.4 Replay Service Runtime Behavior

*Replay Service* currently does not delete *DataWriters*, even if all original *DataWriters* were deleted in the recorded database.

### 4.2 Configuration

When you start *Replay Service*, you can specify a configuration file in XML format. In that file, you can specify the properties that control the behavior of the service. This section describes how to write a configuration file.

#### 4.2.1 How to Load the XML Configuration

*Replay Service* loads its XML configuration from multiple locations. Here are the various sources of configuration files, listed in load order:

- [%working directory%]/USER_REPLAY_SERVICE.xml This file is loaded automatically if it exists.
- [%NDDSHOME%]/resource/xml/RTI_REPLAY_SERVICE.xml This file is loaded automatically if it exists.
- File specified using the command-line parameter `-cfgFile`.

**Note:** [%working directory%] indicates the path to the current working directory from which you run the *Replay Service*.

[%NDDSHOME%] indicates the path to your *Connext DDS* installation (see Section 1.3).

You may use a combination of the above sources and load multiple configuration files.

Here is an example configuration file. You will learn the meaning of each line as you read the rest of this section.
<xml version="1.0" encoding="UTF-8"?>
<!-- All of our files start with a dds tag -->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../schema/rti_replay_service.xsd">
  <!-- Qos Library -->
  <qos_library name="MyQosLib">
    <qos_profile name="DurableQos">
      <datawriter_qos>
        <reliability>
          <kind>RELIABLE_RELIABILITY_QOS</kind>
        </reliability>
      </datawriter_qos>
    </qos_profile>
  </qos_library>

  <!-- Replay service instance -->
  <replay_service name="MyReplayService">
    <!-- Input file to use when replaying -->
    <storage>
      <sqlite>
        <database_dir>.</database_dir>
      </sqlite>
    </storage>

    <!-- Select the begin and end time of the data to replay -->
    <data_selection>
      <time_range>
        <begin_time>
          <sec>200</sec>
          <nanosec>2000000</nanosec>
        </begin_time>
      </time_range>
    </data_selection>

    <!-- Domain Participant for replaying data -->
    <domain_participant name="Participant0">
      <domain_id>0</domain_id>
    </domain_participant>

    <!-- The session that is replaying the data. Contains configuration for publishers, threading, group of topics to replay. -->
    <session name="Session" default_participant_ref="Participant0">
      <!-- Topics to replay in this session -->
      <topic_group name="RecordAll">
        <!-- Topics to replay -->
        <allow_topic_name_filter>*</allow_topic_name_filter>
      </topic_group>
    </session>
  </replay_service>
</dds>
4.2.2 XML Syntax and Validation

The XML representation of DDS-related resources must follow these syntax rules:

- It shall be a well-formed XML document according to the criteria defined in clause 2.1 of the Extensible Markup Language standard.
- It shall use UTF-8 character encoding for XML elements and values.
- It shall use `<dds>` as the root tag of every document.

To validate the loaded configuration, Replay Service relies on an XSD file that describes the format of the XML content. We recommend including a reference to this document in the XML file that contains the service’s configuration; this document provides helpful features in code editors such as Visual Studio®, Eclipse®, and NetBeans®, including validation and auto-completion while you are editing the XML file.

The XSD definitions of the XML elements are in `$NDDSHOME/resource/schema/rti_replay_service.xsd`. (See Section 1.3 for the path to NDDSHOME.)

To include a reference to the XSD document in your XML file, use the attribute `xsi:noNamespaceSchemaLocation` in the `<dds>` tag. For example:

```xml
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../schema/rti_recording_service.xsd">
  <!-- ... -->
</dds>
```

4.2.3 XML Tags for Configuring Replay Service

This section describes the XML tags you can use in a Replay Service configuration file. Figure 4.1 and Table 4.2 describe the top-level tags allowed within the root `<dds>` tag.

Table 4.2: Top-Level Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;dds&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;qos_library&gt;</code></td>
<td>Specifies a QoS library and profiles. The contents of this tag are specified in the same manner as for a Connext DDS QoS profile file — see Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..*</td>
</tr>
<tr>
<td><code>&lt;types&gt;</code></td>
<td>Defines types that can be used by Replay Service. This tag is needed if data types are not available through discovery, or when using a transformation. The type description is done using the Connext DDS XML format for type definitions. See Creating User Data Types with Extensible Markup Language (XML), in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..*</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.2 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;dds&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;plugin_library&gt;</td>
<td>Contains a list of libraries that can be used to:</td>
<td>0..*</td>
</tr>
<tr>
<td></td>
<td>• Plug in custom storage, such as custom databases.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For more information, see Section 4.5.3.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Transform data after it is received from Connext DDS and before it is placed in storage. For more information, see Data Transformation, in the RTI Routing Service User’s Manual.</td>
<td></td>
</tr>
<tr>
<td>&lt;replay_service&gt;</td>
<td><strong>Required.</strong> Specifies a <em>Replay Service</em> configuration. See Section 4.2.4.</td>
<td>1..*</td>
</tr>
<tr>
<td>Attributes</td>
<td>• <strong>name</strong>: uniquely identifies a service configuration. Required.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>&lt;replay_service name=&quot;ReplayAll&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;!-- your service settings ... --&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/replay_service&gt;</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2.4 Replay Service Tag

A configuration file must have at least one `<replay_service>` tag. This tag is used to configure an execution of *Replay Service*.

A configuration file may contain multiple `<replay_service>` tags. When you start *Replay Service*, you can specify which `<replay_service>` tag to use to configure the service using the `-cfgName` command-line parameter. This means one file can be used to configure multiple *Replay Service* executions.

Figure 4.2 and Table 4.3 describe the tags allowed within a `<replay_service>` tag.

Table 4.3: Replay Service Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;replay_service&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;administration&gt;</td>
<td>Enables remote administration. When administration is enabled, monitoring is also enabled by default. See Section 4.2.5.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;monitoring&gt;</td>
<td>Enables monitoring for <em>Replay Service</em>, including statistics. See Section 4.2.6.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;storage&gt;</td>
<td>Describes how the data will be loaded from storage. See Section 4.2.7. If this is not specified, data will be loaded from the current working directory.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page

4.2. Configuration 60
### Table 4.3 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within <code>&lt;re-play_service&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;playback&gt;</code></td>
<td>Specifies the timing rules for how data is played back. See Section 4.2.13.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;data_selection&gt;</code></td>
<td>Enables selection of a subset of data to replay. Supports selecting replay data by time (or tagged time). See Section 4.2.14.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| `<domain_participant>`          | **Required.** Specifies a DomainParticipant to use to replay data. The domain ID specified for the DomainParticipant will be used to determine which domains to select from the database and (combined with the domainIdBase) which domains to replay the data into. For example, if data was originally recorded in domains 0 and 1, and you want to replay in domains 50 and 51, you must:  
  - Specify `<domain_participant>` tags with `<domain_id>` tags set to 0 and 1 in your configuration file.  
  - Use `-domainIdBase 50` to specify that the domain IDs the data will actually be written into is offset by 50.  
  See Section 4.2.12.  
  **Attributes:**  
  - `name`: Uniquely defines a DomainParticipant.  
    **Required.**  
  **Example**  
  ```xml  
  <domain_participant name="Participant3">  
  <domain_id>3</domain_id>  
  <!-- Participant QoS -->  
  </domain_participant>  
  ``` | 1..*          |

Continued on next page
Table 4.3 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;replay_service&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <session>                   | **Required.** Active component of Replay Service for replaying data. Contains one or more threads that can be used for replay. See Section 4.2.16. **Attributes**  
  - name: Uniquely defines a replay session. **Required.**  
  - default_participant_ref: Specifies a default DomainParticipant to be used by topics and topic groups belonging to this replay session. Children can override this by specifying their own participant. **Example**  
  
  ```xml
  <session name="Session" default_participant_ref="Participant3">  
    <!-- ... topics / groups of topics to record -->  
  </session>
  ```
| 1..*                        | 1..*        |

**Example:** Specify a Replay Service Configuration in XML

```xml
<dds>

  <replay_service name="MyReplayService">
    <!-- ... Required entities -->
  </replay_service>

</dds>
```

Starting a Replay Service with the following command will use the `<replay_service>` tag with the name “MyReplayService”.

```
$NDDSHOME/bin/rtireplayservice -cfgFile file.xml -cfgName MyReplayService
```

Replay Service uses the Topic names and domain IDs specified in the configuration file to determine which data to replay. To replay data into a different domain than the one it was recorded from, use the `-domainIdBase` command-line parameter to change the output domain ID.

### 4.2.5 Administration

The `<administration>` tag allows you to enable and configure remote administration of Replay Service, including stopping, starting, and pausing replay.

See Section 4.3 for details on using remote administration.
Figure 4.1: Top-level Tags in the *Replay* Configuration File
Figure 4.2: Tags used to configure a *Replay Service* instance
The domain ID specified inside a DomainParticipant, in combination with the Topic name, is used to load data from recorded tables. In this case, the DomainParticipant’s domain ID is set to 3. The `topic_group` specifies all topics.

To replay data in a different domain than where it was recorded, use the `-domainIdBase` command-line parameter to specify a base ID for the replay domain. Here, by specifying a domainIdBase of 2, all Topics will be replayed in their original domain, plus 2.
### Table 4.4: Administration Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;administration&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;domain_id&gt;</code></td>
<td>Domain ID used for remote administration. Also used for monitoring by default.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;participant_qos&gt;</code></td>
<td>QoS used by the administration DomainParticipant. If the tag is not defined, Connext DDS defaults will be used.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;publisher_qos&gt;</code></td>
<td>QoS used by the administration Publisher. If the tag is not defined, Connext DDS defaults will be used.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;subscriber_qos&gt;</code></td>
<td>QoS used by the administration Subscriber. If the tag is not defined, Connext DDS defaults will be used.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| `<datawriter_qos>`             | QoS used by administration DataWriter(s). If the tag is not defined, Connext DDS defaults will be used, with the following changes:  
  - `history.kind = DDS_KEEP_ALL_HISTORY_QOS`  
  - `resource_limits.max_samples = 32` | 0..1          |
| `<datareader_qos>`             | Quality of Service (QoS) used by administration DataReader(s). If the tag is not defined, the Connext DDS defaults will be used, with the following changes:  
  - `reliability.kind = DDS_RELIABLE_RELIABILITY_QOS` (this value cannot be changed)  
  - `history.kind = DDS_KEEP_ALL_HISTORY_QOS`  
  - `resource_limits.max_samples = 32` | 0..1          |
| `<distributed_logger>`         | When you enable Distributed Logger, Replay Service will publish its Log messages to Connext DDS. See Section 3.2.17. | 0..1          |

The contents of the tags for configuring QoS are specified in the same manner as for the Connext DDS QoS profile file. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.

### 4.2.6 Monitoring

The `<monitoring>` tag allows you to enable and configure remote monitoring of Replay Service.

### Table 4.5: Monitoring Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;monitoring&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;enabled&gt;</code></td>
<td>Whether to enable monitoring of the service. Default: Disabled unless administration is enabled</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.5 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within <code>&lt;monitoring&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;domain_id&gt;</code></td>
<td>Domain ID used for monitoring. Default: Use the domain ID specified for monitoring</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;datawriter_qos&gt;</code></td>
<td>QoS used by monitoring DataWriter(s)</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;publisher_qos&gt;</code></td>
<td>QoS used by monitoring Publisher(s)</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;participant_qos&gt;</code></td>
<td>QoS used by monitoring DomainParticipant</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;statistics_sampling_period&gt;</code></td>
<td>How frequently to sample the service’s statistics, using the tags <code>&lt;sec&gt;</code> or <code>&lt;nanosec&gt;</code>. For example, <code>&lt;sec&gt;1&lt;/sec&gt;</code> samples the service’s statistics every second.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;status_publication_period&gt;</code></td>
<td>How frequently to publish the service status, using the tags <code>&lt;sec&gt;</code> or <code>&lt;nanosec&gt;</code>. For example, <code>&lt;sec&gt;1&lt;/sec&gt;</code> publishes the service’s status every second.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

The contents of the tags for configuring QoS are specified in the same manner as for the Connext DDS QoS profile file. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.

4.2.7 Storage

The `<storage>` tag allows you to configure the storage from which data will be read. You can choose between using the built-in SQLite storage or implementing your own storage plugin.

Table 4.6: Storage Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;storage&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;sqlite&gt;</code></td>
<td>Enables replaying data from an SQLite database file See Section 4.2.8.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;plugin&gt;</code></td>
<td>Enables replaying data from an external library that you specify. See Section 4.2.9 for more about using this tag. See Section 4.5.3 for a tutorial on plugging in custom storage. Attributes • plugin_name: The name of the plugin to use from the plugin_library. This is the name of the plugin to load, qualified by the plugin library name.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;legacy&gt;</code></td>
<td>Enables replaying data from a legacy (pre-6.0.0) database file. See Section 4.2.10.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

4.2.8 SQLite

The `<sqlite>` tag allows you to specify a SQLite formatted file to read data from for replay.
Table 4.7: SQLite Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;sqlite&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;storage_format&gt;</td>
<td>The storage format of the user data files found in the database directory. If not specified, XCDR_AUTO is assumed.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td>The options are:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• XCDR_AUTO: This is the binary format used by Connext DDS when sending data over the network. This has the highest performance for recording, but can only be viewed by using Converter to convert the data to a readable format, or by using Replay to replay the data. This will internally store data in XCDR or XCDR2 depending on the format received.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• JSON_SQLITE: This format can be queried, but recording in this format has lower performance because data must be deserialized before it can be stored.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• XCDR: The format to use when communicating with Connext DDS before 6.0.0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• XCDR2: More efficient than XCDR, used by Connext DDS 6.0.0 and later.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: XCDR_AUTO.</td>
<td></td>
</tr>
<tr>
<td>&lt;database_dir&gt;</td>
<td>The directory where to look for a recorded database. The directory must contain a valid metadata file, as well as a discovery file and one or many user data files. This field is optional. When not included, the current directory will be used. Default: the current directory.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.7 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;sqlite&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sql_initialization_string&gt;</td>
<td>Allows configuring an SQLite SQL expression to use when establishing SQLite connections using this plugin. You can add an index to a table to speed up replay, but this must be done for each table you want to index. Note that you can perform this step using an SQLite client.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$&lt;$sql_initialization_string$&gt;$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CREATE INDEX IF NOT EXISTS pingtopic_0_by_rcp_timestamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ON 'PingTopic0' (SampleInfo_reception_timestamp);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$&lt;$sql_initialization_string$&gt;$</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> when using Recorder and Replay at the same time for the same DB file(s), or for that matter, any other SQLite application accessing the data, we recommend using SQLite’s WAL (write-ahead logging) mode. This can be done by adding <code>PRAGMA JOURNAL_MODE = WAL;</code> to this configuration setting. More information about SQLite’s WAL mode can be found <a href="#">here</a>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Default: `PRAGMA SYNCHRONOUS = OFF; PRAGMA JOURNAL_MODE = MEMORY;` | | |}

### 4.2.9 Plugin

This tag allows you to specify a plugin library that you can use to replay data from storage of your choice.

See Section 4.5.3 for a tutorial on plugging in custom storage.

Table 4.8: Plugin Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;plugin&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;property&gt;</td>
<td>Name/value pairs of properties to pass to a storage plugin.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

### 4.2.10 Legacy

Table 4.9: Legacy Tags in the Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;legacy&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;file_path&gt;</td>
<td>Path to the legacy format recorded file. File set and version properties will be obtained automatically from the file itself.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;domain_mapping&gt;</td>
<td>This tag allows you to link legacy domain names with domain IDs. See Section 4.2.11.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
4.2.11 Domain Mapping

When converting a legacy database, the domain may not have been recorded. This tag provides a way to map a table with a domain ID.

Later versions of the old Recorder database allowed you to use field filters. By default, the domain ID field was not recorded (it was filtered out by default). Thus, there is no information available in these legacy databases to relate the domain name used to record the data with a domain ID. This tag allows you to link legacy domain names with domain IDs.

<table>
<thead>
<tr>
<th>Tags within &lt;domain_mapping&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;domain_map&gt;</td>
<td>Required. A link between a recorded legacy domain name and a domain ID.</td>
<td>1..*</td>
</tr>
</tbody>
</table>

Attributes:
- legacy_domain_name: Name of the recorded domain name as specified in the old Recording Service domain tag, for example: <domain name="domain0">.
- participant_ref: The name of a DomainParticipant specified below using a <domain_participant> tag. This DomainParticipant will be used to replay data specified by the legacy domain name.

4.2.12 DomainParticipant

The <domain_participant> tag allows you to specify the DomainParticipants and domain IDs you want to query from the database for replay. You can replay in the same domain you used to record, or you can use the -domainIdBase command-line parameter to replay in a different domain.

The contents of the tags for configuring QoS are specified in the same manner as for the Connext DDS QoS profile file. See Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.

<table>
<thead>
<tr>
<th>Tags within &lt;domain_participant&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;domain_id&gt;</td>
<td>The domain ID of tables loaded from the database. This is used (in conjunction with the -domainIdBase command-line parameter) to choose which domain to replay into.</td>
<td>1</td>
</tr>
<tr>
<td>&lt;participant_qos&gt;</td>
<td>QoS used by this DomainParticipant.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.11 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;domain_participant&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <memory_management>              | Configures certain aspects of how Connext DDS allocates internal memory. The configuration is per DomainParticipant and therefore affects all the contained DDS entities. **Example:**<br>  
  
  `<memory_management>`<br>  
  `<sample_buffer_min_size>`<br>  
  1024  
  `</sample_buffer_min_size>`<br>  
  `<sample_buffer_trim_to_size>`<br>  
  true  
  `</sample_buffer_trim_to_size>`  
  `</memory_management>`  
  | This tag includes the following tags:  
  | • **sample_buffer_min_size**: For all DataWriters and DataReaders, the way Connext DDS allocates memory for samples is as follows: Connext DDS pre-allocates space for samples up to size X in the DataWriter and DataReader queues. If a sample has an actual size greater than X, the memory is allocated dynamically for that sample. The default size is 64KB. This is the maximum amount of pre-allocated memory. Dynamic memory allocation may occur when necessary if samples require a bigger size.  
  | • **sample_buffer_trim_to_size**: If set to true, after allocating dynamic memory for very large samples, that memory will be released when possible. If false, that memory will not be released but kept for future samples if needed. The default is false.  
  
  This feature is useful when a data type has a very high maximum size (e.g., megabytes) but most of the samples sent are much smaller than the maximum possible size (e.g., kilobytes). In this case, the memory footprint is reduced dramatically, while still correctly handling the rare cases in which very large samples are published. | 0..1 |
| <register_type>                  | Registers a type name and associates it with a type representation. When you define a type in the configuration file, you have to register the type in order to use it in a <topic>. | 0..* |

4.2.13 Playback

The `<playback>` tag allows you to specify the timing rules for how the data is played back, such as the time on the local machine to start the replay, if it is not started immediately.
Table 4.12: Playback Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;playback&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;start_replay_local_time&gt;</td>
<td>Selects the local time when <em>Replay Service</em> should start publishing samples. Specified in hours, minutes, and seconds. Specifying a full date is not currently supported, so if you specify a time that happened in the past (which <em>Replay Service</em> interprets as in the past <em>today</em>), replay will start immediately. Default: Replay will start immediately</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;fidelity&gt;</td>
<td>Specifies the fidelity of the replayed data. Samples whose timestamp distance is less than the fidelity period may be published together in the same batch. For example, if your fidelity is 100 ms, data stored in your database within a 100 ms time period may be published together.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;rate&gt;</td>
<td>Specifies the rate to replay data, as a double. For example, to play data at half speed, use 0.5.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;enable_looping&gt;</td>
<td>Plays the data in a repeating loop, until <em>Replay Service</em> is stopped. Note that this means that only a termination signal or a remote shutdown command will make it stop replaying data. When no time range data selection has been specified (see Section 4.2.14) then Replay will replay the whole contents of the database file(s) and when it’s time to loop, it will start from the very beginning of the file or files. However, when a specific time range has been specified for replay, the looping will respect the time range and will restart at the specified start time and end at the specified end time. <strong>Note:</strong> in our default SQLite implementation, Replay is able to check if new data has been written by Recorder (or any application) for every given stream being replayed, since the start of the service. This feature is generally not supported when looping without specifying a time range and may yield unexpected results.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

4.2.14 Data Selection

Selection of data to replay from the database. Currently only supports selection using a time range.

Table 4.13: Data Selection Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;data_selection&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;time_range&gt;</td>
<td>Select data to replay from the database, based on start and end times. See Section 4.2.15.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
4.2.15 Time Range

The `<time_range>` tag allows you to specify the begin and end times of the data you want to replay. This can be specified either as timestamps or as symbolic timestamps called “timestamp tags.” These tags may have been added to the recording through remote administration, and can be viewed using the script `rtirecordingservice_list_tags` (see Section 3.5.7).

<table>
<thead>
<tr>
<th>Tags within <code>&lt;time_range&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;begin_time&gt;</code></td>
<td>Select data to replay from the database, with a timestamp beginning at this time. Specify the time in seconds and nanoseconds. Default: 0, start at beginning of the file.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;begin_tag&gt;</code></td>
<td>Can be used instead of begin_time. Select data to start replaying from the database by specifying the string name of a timestamp tag. Timestamp tags associate a timestamp with a name. Then you can refer to a timestamp by that name. The timestamp must have been tagged with that name during recording. See Section 3.5.6. Default: Start at beginning of the file.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;end_time&gt;</code></td>
<td>Select data to replay from the database, with a timestamp ending at this time. Specify the time in seconds and nanoseconds. Default: Stop at end of file.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;end_tag&gt;</code></td>
<td>Can be used instead of end_time. Select when to stop replaying data by specifying the string name of a timestamp tag. Timestamp tags associate a timestamp with a name. Then you can refer to a timestamp by that name. The timestamp must have been tagged with that name during recording. See Section 3.5.6. Default: Stop at end of file.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

4.2.16 Session

The `<session>` tag configures the threads that will be used to replay data. You also specify the Topics and groups of Topics to replay inside the `<session>` tag.

<table>
<thead>
<tr>
<th>Tags within <code>&lt;session&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;publisher_qos&gt;</code></td>
<td>Specifies the QoS of the Publisher that will be used by the contained <code>&lt;topic&gt;</code> and <code>&lt;topic_group&gt;</code>. For information on configuring Publisher QoS with XML, see Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.15 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within <code>&lt;session&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;thread_pool&gt;</code></td>
<td>Defines the number of threads used by this session to process Topics and Topic Groups and sets the mask, priority, and stack size of each thread.</td>
<td>0..1</td>
</tr>
<tr>
<td></td>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;thread_pool&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;mask&gt;MASK_DEFAULT&lt;/mask&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;priority&gt;THREAD_PRIORITY_DEFAULT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/priority&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;stack_size&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>THREAD_STACK_SIZE_DEFAULT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/stack_size&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/thread_pool&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Default values:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• size: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• mask: MASK_DEFAULT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• priority: THREAD_PRIORITY_DEFAULT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• stack_size: THREAD_STACK_SIZE_DEFAULT</td>
<td></td>
</tr>
<tr>
<td><code>&lt;topic&gt;</code></td>
<td>Specifies an individual Topic to replay. See Section 4.2.18. <strong>Attributes:</strong></td>
<td>0..*</td>
</tr>
<tr>
<td></td>
<td>• name: The name of the Topic to replay. This name is also used when monitoring and administering each Topic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• participant_ref: A DomainParticipant to use when replaying this Topic. If the parent <code>&lt;session&gt;</code> specifies a default_participant_ref, this attribute is optional.</td>
<td></td>
</tr>
<tr>
<td><code>&lt;topic_group&gt;</code></td>
<td>Specifies a group of Topics to replay. See Section 4.2.17. <strong>Attributes:</strong></td>
<td>0..*</td>
</tr>
<tr>
<td></td>
<td>• name: The name of the topic group. This name is used when monitoring and administering each topic group.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• participant_ref: A DomainParticipant to use when replaying this topic group. If the parent <code>&lt;session&gt;</code> specifies a default_participant_ref, this attribute is optional.</td>
<td></td>
</tr>
</tbody>
</table>
4.2.17 Topic Group

The `<topic_group>` tag allows you to replay a group of Topics, using regular expressions to describe which Topics to replay.

Table 4.16: Topic Group Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;topic_group&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| `<allow_topic_name_filter>` | A regular expression describing which Topics are allowed to be replayed. **Example:**  
`<topic_group name="ReplayAll">  
  <allow_topic_name_filter>*</allow_topic_name_filter>  
</topic_group>` | 0..1 |
| `<deny_topic_name_filter>` | A regular expression describing which Topics are not allowed to be replayed. This is applied after the `<allow_topic_name_filter>`. | 0..1 |
| `<allow_type_name_filter>` | A regular expression describing the names of data types that are allowed to be replayed. | 0..1 |
| `<deny_type_name_filter>` | A regular expression describing the names of data types that are not allowed to be replayed. This is applied after the `<allow_type_name_filter>`. | 0..1 |
| `<datawriter_qos>` | The DataWriter’s QoS to use when replaying this data. For information on configuring QoS with XML, see Configuring QoS with XML, in the *RTI Connext DDS Core Libraries User’s Manual*. | 0..1 |

4.2.18 Topic

The `<topic>` tag specifies an individual Topic to replay.

Table 4.17: Topic Tags in Replay Service’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;topic&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;registered_type_name&gt;</code></td>
<td>The name of the data type that will be replayed for this topic. <strong>Required.</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 4.17 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| `<transformation>` | The transformation library to be applied to this Topic’s data when replaying. This is a user library that can modify the data after it is received from storage and before it is sent via Connext DDS. Transformations implement APIs identical to Routing Service’s transformations. For more on using transformations, see these sections in the RTI Routing Service User’s Manual:  
  - Data Transformation  
  - Tutorials  
**Attributes:**  
  - `plugin_name`: The name of the plugin to load, qualified by the plugin library name.  
**Example:**  
```xml  
<dds>  
  <plugin_library name="ReplayTransformations">  
    <transformation_plugin name="ModifyTestID">  
      <create_function>ModifyTestID_create</create_function>  
      <dll>modify_test_id_library</dll>  
    </transformation_plugin>  
  </plugin_library>  
</dds>  
```
| 0..1 |

| `<datawriter_qos>` | The DataWriter QoS to use when replaying this data. For information on configuring QoS with XML, see Configuring QoS with XML, in the RTI Connext DDS Core Libraries User’s Manual. | 0..1 |

### 4.3 Remote Administration

This section provides documentation on Replay Service remote administration. A control client (such as RTI Admin Console) can use this interface to remotely control a Replay Service.

**Note:** Replay Service remote administration is based on the RTI Remote Administration Platform described in Section 6.2. Please refer to that manual for a detailed discussion on the workings of remote administration in Replay Service.

Below you will find an API reference for all the supported operations.
4.3.1 Enabling Remote Administration

By default, remote administration is disabled in *Replay Service*. To enable remote administration you can use the `<administration>` XML tag (see Section 4.2) or the `-remoteAdministrationDomainId` command-line parameter (see Section 4.1). Both of these methods enable remote administration and set the domain ID for remote communication.

4.3.2 Available Service Resources

Table 4.18 lists the public resources specific to *Replay Service*. Each resource identifier is expressed as a hierarchical sequence of identifiers, including parent and target resources. (See Section 6.1.1 for details.)

In the table below, the elements (rs) and (st) refer to the name of an entity of the corresponding class as specified in the configuration in the `name` attribute. For example, in the following configuration:

```xml
<replay_service name="MyReplay">...</replay_service>
```

The resource identifier is:

```
/replay_service/MyReplay
```

In the table, the resource identifier is written as `/replay_service/(rs)`, where `(rs)` is the service name. This nomenclature is used in the table to give you an idea of the structure of the resource identifiers. For actual (example) resource identifier names, see the example section that follows.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Resource Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replay Service</td>
<td>/replay_service/(rs)</td>
</tr>
</tbody>
</table>

**Example**

This example shows you how to address a resource of each possible resource class in *Replay Service*.

**Replay Service**

Entity with name “MyReplay”:

```xml
<replay_service name="MyReplay">...</replay_service>
```

Resource identifier:

```
/replay_services/MyReplay
```
4.3.3 Remote API Overview

Table 4.19: Remote Interface Overview

<table>
<thead>
<tr>
<th>Re-source</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replay-Service</td>
<td>DELETE /replay_services/(rs)</td>
<td>Shuts down a running Replay Service</td>
</tr>
<tr>
<td></td>
<td>UPDATE /replay_services/(rs)/state</td>
<td>Sets a Replay Service state</td>
</tr>
</tbody>
</table>

4.3.4 Replay Service

DELETE /replay_services/(rs)

Operation shutdown

Causes Replay Service to shutdown.

UPDATE /replay_services/(rs)/state

Operation set state

See Set Resource State (Section 6.2.3).

Valid requested states:

- STARTED
- STOPPED
- PAUSED
- RUNNING

Example

To pause a replay service with the name “MyReplay”:

<table>
<thead>
<tr>
<th>Request Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_action</td>
<td>UPDATE</td>
</tr>
<tr>
<td>resource_identifier</td>
<td>/replay_services/MyReplay/state</td>
</tr>
<tr>
<td>octet_body</td>
<td>to_cdr_buffer(RTI::Service::EntityStateKind::PAUSED)</td>
</tr>
</tbody>
</table>

4.4 Monitoring

This section provides documentation on Recording Service remote monitoring.

Note: Recording Service monitoring is based on the Monitoring Distribution Platform described in Section 6.3. We recommend that you read Section 6.3 before using Recording Service monitoring.
4.4.1 Overview

Enabling Service Monitoring

By default, monitoring is disabled in *Recording Service*. To enable monitoring you can use the `<monitoring>` tag (see Section 3.2.5) or the `-remoteMonitoringDomainId` command-line parameter, which enables remote monitoring and sets the domain ID for data publication (see Section 3.1.3).

Monitoring Types

The available *Keyed Resource* classes and their types that can be present in the distribution monitoring topics are listed in Table 4.20. The complete type relationship is shown in Figure 4.5.

<table>
<thead>
<tr>
<th>Keyed Resource Class</th>
<th>Config</th>
<th>Event</th>
<th>Periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>ServiceConfig</td>
<td>ServiceEvent</td>
<td>ServicePeriodic</td>
</tr>
<tr>
<td>Session</td>
<td>SessionConfig</td>
<td>SessionEvent</td>
<td>SessionPeriodic</td>
</tr>
<tr>
<td>TopicGroup</td>
<td>TopicGroupConfig</td>
<td>TopicGroupEvent</td>
<td>TopicGroupPeriodic</td>
</tr>
<tr>
<td>Topic</td>
<td>TopicConfig</td>
<td>TopicEvent</td>
<td>TopicPeriodic</td>
</tr>
</tbody>
</table>

Figure 4.5: Keyed Resource Types for *Recording Service* monitoring

All the type definitions for *Recording Service* monitoring information are in `[NDDSHOME]/resource/idl/ServiceCommon.idl` and `[NDDSHOME]/resource/idl/RecordingServiceMonitoring.idl`. *Recording Service* creates a *DataWriter* for each distribution *Topic*. All *DataWriters* are created
from a single Publisher, which is created from a dedicated DomainParticipant. See Section 3.2.5 for details on configuring the QoS for these entities.

## 4.4.2 Monitoring Metrics Reference

This section provides a reference to all the monitoring metrics Recording Service distributes, organized by service resource class.

### Service

**Listing 4.1: Recording Service Types**

```cpp
@mutable @nested
struct SqliteDatabaseConfig {
    Service::FilePath db_directory;
    @optional Service::FilePath execution_directory_expression;
    @optional Service::FilePath user_data_file_expression;
};

@mutable @nested
struct SqliteDatabaseEvent {
    @optional Service::FilePath current_db_directory;
    @optional Service::FilePath current_file;
    @optional int32 rollover_count;
};

@mutable @nested
struct SqliteDatabasePeriodic {
    @optional Service::FilePath current_file;
    @optional uint64 current_file_size;
    int32 current_timestamp_sec;
    uint32 current_timestamp_nanosec;
};

@mutable @nested
struct ParticipantInfo {
    Service::BoundedString name;
};

@mutable @nested
struct ServiceConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString application_name;
    Service::Monitoring::ResourceGuid application_guid;
    @optional Service::Monitoring::HostConfig host;
    @optional Service::Monitoring::ProcessConfig process;
    @optional SqliteDatabaseConfig builtin_sqlite;
    @optional sequence<ParticipantInfo> participants;
};

@mutable @nested
struct ServiceEvent : Service::Monitoring::EntityEvent {
    //to avoid unused variable warnings
    @optional SqliteDatabaseEvent builtin_sqlite;
};
```

(continues on next page)
struct ServicePeriodic {
    @optional Service::Monitoring::HostPeriodic host;
    @optional Service::Monitoring::ProcessPeriodic process;
    @optional SqliteDatabasePeriodic builtin_sqlite;
};

Table 4.21: ServiceConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>application_name</td>
<td>Name of the Recording Service instance. The application name is provided through:</td>
</tr>
<tr>
<td></td>
<td>• appName command-line option when run as executable.</td>
</tr>
<tr>
<td></td>
<td>• ServiceProperty::application_name field when run as a library.</td>
</tr>
<tr>
<td>application_guid</td>
<td>GUID of the Recording Service instance. Unique across all service instances.</td>
</tr>
<tr>
<td>host</td>
<td>See Table 6.9.</td>
</tr>
<tr>
<td>process</td>
<td>See Table 6.11.</td>
</tr>
<tr>
<td>builtin_sqlite</td>
<td>See Table 4.22</td>
</tr>
<tr>
<td>participants</td>
<td>Sequence of ParticipantInfo objects, one for each DomainParticipant inside the Recording Service. See Table 4.23.</td>
</tr>
</tbody>
</table>

Table 4.22: SqliteDatabaseConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db_directory</td>
<td>Path to the base directory where the database files live.</td>
</tr>
<tr>
<td>execution_directory_expression</td>
<td>This value is not set when running Replay Service. See Section 3.2.10</td>
</tr>
<tr>
<td>user_data_file_expression</td>
<td>This value is not set when running Replay Service.</td>
</tr>
</tbody>
</table>

Table 4.23: ParticipantInfo

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the DomainParticipant instance, as specified in the name attribute of the corresponding configuration tag.</td>
</tr>
</tbody>
</table>
Table 4.24: ServiceEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
<tr>
<td>builtin_sqlite</td>
<td>See Table 4.25</td>
</tr>
</tbody>
</table>

Table 4.25: SqliteDatabaseEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_directory</td>
<td>Path to the current directory where files are being replayed from.</td>
</tr>
<tr>
<td>current_file</td>
<td>This value is not set when running <em>Replay Service</em></td>
</tr>
<tr>
<td>rollover_count</td>
<td>This value is not set when running <em>Replay Service</em></td>
</tr>
</tbody>
</table>

Table 4.26: ServicePeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>See Table 6.10.</td>
</tr>
<tr>
<td>process</td>
<td>See Table 6.12.</td>
</tr>
<tr>
<td>builtin_sqlite</td>
<td>See Table 4.27</td>
</tr>
</tbody>
</table>

Table 4.27: SqliteDatabasePeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_file_size</td>
<td>This value is not set when running <em>Replay Service</em></td>
</tr>
<tr>
<td>current_timestamp_sec</td>
<td>The seconds part of the timestamp at which data is being replayed, relative to recorded time.</td>
</tr>
<tr>
<td>current_timestamp_nanos</td>
<td>The nanoseconds part of the timestamp at which data is being replayed, relative to recorded time.</td>
</tr>
</tbody>
</table>

Session

Listing 4.2: *Session* Types

```cpp
@mutable @nested
struct SessionConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString default_participant_name;
};

@mutable @nested
struct SessionEvent : Service::Monitoring::EntityEvent {
    //to avoid unused variable warnings
    int32 _dummy;
};

@mutable @nested
struct SessionPeriodic {
    @optional Service::Monitoring::NetworkPerformance network_performance;
}
```
(continues on next page)
Table 4.28: SessionConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>default_participant_name</td>
<td>The name of the default participant configuration.</td>
</tr>
</tbody>
</table>

Table 4.29: SessionEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
</tbody>
</table>

Table 4.30: SessionPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_performance</td>
<td>Provides network performance metric as an aggregation of the same metric across the contained Topic and TopicGroups. See Section 6.3.4.</td>
</tr>
</tbody>
</table>

**TopicGroup**

Listing 4.3: *TopicGroup* Types

```cpp
@mutable @nested
struct TopicGroupConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString participant_name;
};
@mutable @nested
struct TopicGroupEvent : Service::Monitoring::EntityEvent {
    //to avoid unused variable warnings
    int32 _dummy;
};
@mutable @nested
struct TopicGroupPeriodic {
    @optional Service::Monitoring::NetworkPerformance network_performance;
    int64 topic_count;
};
```
Table 4.31: TopicGroupConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>participant_name</td>
<td>Name of the DomainParticipant from which the Topic is created.</td>
</tr>
</tbody>
</table>

Table 4.32: TopicGroupEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
</tbody>
</table>

Table 4.33: TopicGroupPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_performance</td>
<td>Provides network performance metric as an aggregation of the same metric across the contained Topic. See Section 6.3.4.</td>
</tr>
<tr>
<td>topic_count</td>
<td>Current number of Topic created from this TopicGroup.</td>
</tr>
</tbody>
</table>

Topic

Listing 4.4: Topic Types

```cpp
@mutable @nested
struct TopicConfig : Service::Monitoring::EntityConfig {
    Service::BoundedString topic_name;
    Service::BoundedString registered_type_name;
    Service::BoundedString participant_name;
    Service::Monitoring::ResourceGuid topic_group;
};
@mutable @nested
struct TopicEvent : Service::Monitoring::EntityEvent {
    //to avoid unused variable warnings
    int32 _dummy;
};
@mutable @nested
struct TopicPeriodic {
    @optional Service::Monitoring::NetworkPerformance network_performance;
    @optional Service::Monitoring::CountStatus matched_status;
};
```
Table 4.34: TopicConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityConfig</td>
<td>See Table 6.13.</td>
</tr>
<tr>
<td>topic_name</td>
<td>Topic name as specified in the configuration.</td>
</tr>
<tr>
<td>registered_type_name</td>
<td>Topic registered type name as specified in the configuration.</td>
</tr>
<tr>
<td>participant_name</td>
<td>Name of the DomainParticipant from which the Topic is created.</td>
</tr>
<tr>
<td>topic_group</td>
<td>GUID of the TopicGroup from which this Topic was created. This field is set to zero for standalone Topic.</td>
</tr>
</tbody>
</table>

Table 4.35: TopicEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited fields from EntityEvent</td>
<td>See Table 6.14.</td>
</tr>
</tbody>
</table>

Table 4.36: TopicPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network_performance</td>
<td>Provides network performance metric as an aggregation of the same metric across the contained Topic. See Section 6.3.4.</td>
</tr>
<tr>
<td>matched_status</td>
<td>Provides information about the matched endpoints associated with this Topic.</td>
</tr>
</tbody>
</table>

4.5 Tutorials

4.5.1 Example: Getting Started with Replay and Shapes Demo

Start by recording Square data, as described in Section 3.5.1.

Edit the Replay Configuration

In your `<RTI_WORKSPACE>/user_config/recording_service` directory, edit the USER_REPLAY_SERVICE.xml file. Change the storage_format to be JSON_SQLITE.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- All of our files start with a dds tag -->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/6.0.0/rti_replay_service.xsd">

  <replay_service name="UserReplayService">
    <storage>
      (continues on next page)
    </storage>
  </replay_service>
</dds>
```
<sqlite>
  <storage_format>JSON_SQLITE</storage_format>
  <!-- No <database_dir> specified means current directory -->
</sqlite>
</storage>

<!-- Optionally select the begin and end times for the data to be replayed -->
<data_selection>
  <time_range>
    <begin_time>
      <sec>0</sec>
      <nanosec>0</nanosec>
    </begin_time>
  </time_range>
</data_selection>

<!-- Specify playback behavior, including what local time to start -->
<playback>
</playback>

<!-- Configuration for the service: DomainParticipant configuration, storage configuration -->
<domain_participant name="DefaultParticipant">
  <domain_id>0</domain_id>
</domain_participant>

<session name="DefaultSession"
  default_participant_ref="DefaultParticipant">
  <!-- Topics to replay in this session -->
  <topic_group name="DefaultTopicGroup">
    <!-- Topics to replay -->
    <allow_topic_name_filter>*</allow_topic_name_filter>
    <deny_topic_name_filter>rti/*</deny_topic_name_filter>
  </topic_group>
</session>
</replay_service>
</dds>

Start Shapes Demo and Subscribe to Squares

Start *Shapes Demo* from the *Launcher* tool and create a Square subscriber as described in Section 3.5.1.

Start Replay Service

Start *Replay Service*:

```
<NDSDHOME>/bin/rtireplayservice -cfgName UserReplayService -verbosity 3
```

You should see Square data replayed in *Shapes Demo*.
4.5.2 Example: Replaying Data at a Different Rate

To replay data at a faster or slower rate than it was recorded, you can edit the configuration file to specify a playback rate.

Start by recording Square data as described in Section 3.5.1.

Edit the Replay Configuration

In your `<RTI_WORKSPACE>/user_config/recording_service` directory, edit the `USER_REPLAY_SERVICE.xml` file. Add a playback rate, as seen in the following XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!- All of our files start with a dds tag -->
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="https://community.rti.com/schema/6.0.0/rti_replay_service.xsd">

<replay_service name="UserReplayServiceJson">
    <!-- This will look for files in the directory 'json_recording' in the current working directory. This is integrated with the Recording Service configuration 'UserRecorderServiceJson' in the file USER_RECORDING_SERVICE.xml -->
    <storage>
        <sqlite>
        </sqlite>
    </storage>
</replay_service>
```

(continues on next page)
Start Shapes Demo
Start *Shapes Demo* from the *Launcher* tool and create a Square subscriber.

Start Replay Service
Start *Replay Service*:

```
<NDDSHOME>/bin/rtireplayservice -cfgName UserReplayService -verbosity 3
```

You should see Square data replayed in *Shapes Demo* twice as fast as it was originally recorded.
4.5.3 Example: Plugging in Custom Storage

If you created a storage plugin to record data into your own custom storage, you can also create a plugin to replay from that same storage.

**Note:** The APIs for custom storage are not available in the Basic bundle.

There are full examples written in C and C++ about plugging in custom storage in *Recording Service*, in the RTI Community Recording Service examples: C storage plugin and RTI Community Recording Service examples: C++ storage plugin.

**Custom Storage API Overview**

To retrieve data for replay, you must implement the following APIs:

- **A create storage reader API:**
  - This is used to create a StorageReader object.
  - This API is a C function. In C++, you can use macros to declare and define the C function for your class. For example:
    * RTI_RECORDING_STORAGE_READER_CREATE_DECL(FileStorageReader)
    * RTI_RECORDING_STORAGE_READER_CREATE_DEF(FileStorageReader)
The StorageReader is used to create and delete StorageStreamInfoReaders and StorageStreamReaders.

**StorageReader**:
- A create stream info reader API, where you create a stream reader that provides information about what streams (topics) are in your storage.
- A delete stream info reader API, where you delete a stream info reader
- A create stream reader API, where you create a stream reader
- A delete stream reader API, where you delete a stream reader

**StorageStreamInfoReader**:
- Read: An API to retrieve discovery data from the storage. It uses a selector object to determine the kind of samples (not read or any kind) to be returned, as well as the time range. The samples should be returned in increasing time-stamp order.
- Return loan: An API that notifies the plugin that it can release resources associated with the data passed to the read API
- Get service start time: An API to query the recorded time from which to start replaying
- Get service stop time: An API to query the recorded time at which to stop replaying
- Finished: An API to tell *Replay Service* and *Converter* that there are no more stream infos to read.
- Reset: An API called by the *Replay Service* to tell the plug-in to reset its state because it is looping. After this method is called, the stream reader should be ready to start reading data from the beginning of the stream, again.

**StorageStreamReader**
- Read: An API to retrieve data from storage. It uses a selector object to determine the kind of samples (not read or any kind) to be returned, as well as the time range. The samples should be returned in increasing time-stamp order.
- Return loan: An API that notifies the plugin that it can release resources associated with the data passed to the take API
- Finished: An API that notifies the plugin that there is no more data in this data stream

The APIs provide a mechanism to have strongly-typed StorageStreamReader classes and a builtin one is provided based on *dds: :core: :xtypes: :DynamicData*.

More detailed API documentation is located in:
- Recording Service C API documentation.
- Recording Service C++ API documentation.
4.5.4 Using Timestamp Tags with Replay Service

If your recording was originally made with the builtin SQLite storage plugin, and you used the `tag_timestamp` remote command to tag certain events, then your recording contains timestamp tags: symbolic timestamp names you can use in place of timestamps expressed in units of time. For more information on timestamp tags, see Section 3.5.6.

You can list the timestamp tags that are in your recorded database by using the `rtirecordingservice_list_tags` script. Use the `-d` argument to point to the directory that contains your recorded database, as follows:

```
<NDDSHOME>/bin/rtirecordingservice_list_tags -d /database/directory/
```

This command will analyze the recording in `/database/directory/` and list the details of any timestamp tags in the recording, including the tag names, descriptions, and associated timestamps.

You can use the `tag_name` of the timestamp tags you find in a recording when you are creating an XML configuration file for Replay Service by using the `<data_selection>` tag.

For example, if after running `rtirecordingservice_list_tags`, you see output such as:

<table>
<thead>
<tr>
<th>tag_name</th>
<th>timestamp_ms</th>
<th>tag_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/my_example/my_events/tag1</td>
<td>1546484663309</td>
<td>first tag description</td>
</tr>
<tr>
<td>/my_example/my_events/tag2</td>
<td>1546484703360</td>
<td>a second tag description</td>
</tr>
</tbody>
</table>

Then you can have a `<data_selection>` tag in your XML for Replay Service, after the `<storage>` tag, that looks like the following:

```
<data_selection>
    <time_range>
        <begin_tag>/my_example/my_events/tag1</begin_tag>
        <end_tag>/my_example/my_events/tag2</end_tag>
    </time_range>
</data_selection>
```

Replay will replay data between those tags. Note that when expressing a `<time_range>` tag, you can mix and match timestamps and timestamp tags. For example, you can use a `<begin_tag>` (by referring to a `tag_name`) to express the time when replay should begin, and an `<end_tag>` with an end time timestamp (expressed in time units) to express when replay should end. If you do not provide one of the bounds, then the start of recording is the default begin bound and the end of recording is the default end bound.

4.6 Troubleshooting

4.6.1 No Input File

If Replay Service is started but there is no file for it to read from, it will print error messages like the ones below and then exit. It is looking for a database located in a cdr_recording directory inside your current working directory. You can fix this by creating a configuration file that specifies the correct location of your database to replay, or by changing directories to a location that contains your cdr_recording directory that contains a recording.
create_connection: caught exception from:
    set_properties: !No valid metadata file found in directory: cdr_recording

ROUTERConnection_enable: (adapter=StorageAdapterPlugin, retcode=0: set_properties: !No valid metadata file found in directory: cdr_recording)

ROUTERDomainRoute_start: !enable Connection

ROUTERService_startDomainRoute: !start domain route

ROUTERService_createDomainRoute: !start domain route

ROUTERService_start: !create domain route

RTI_RoutingService_start: !start routing service

main: !!start RoutingService error
Chapter 5

Converter

5.1 Usage

This section explains how to run *Converter* from a command line. In particular, it describes:

- How to Start *Converter* (Section 5.1.1).
- *Converter* command-line parameters (Section 5.1.2).

5.1.1 Starting Converter

*Converter* runs as a separate application. The script to run the executable is in `<NDDSHOME>/bin`. (See Section 1.3 for the path to NDDSHOME.)

```
rticonverter [options]
```

To start *Converter* with a default configuration, enter:

```
$NDDSHOME/bin/rticonverter
```

Note: *Converter* is pre-loaded with a builtin configuration that has default settings. See Section 5.2.20.

5.1.2 Converter Command-Line Parameters

The following table describes all the command-line parameters available in *Converter*. To list the available parameters, run `rticonverter -help`.

All command-line parameters are optional; if specified, they override the values of any corresponding settings in the loaded XML configuration. See Section 5.2.4 for the XML elements that can be overridden with command-line parameters.
Table 5.1: Converter Command-Line Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cfgFile &lt;string&gt;</td>
<td>Semicolon-separated list of configuration file paths. <strong>Default:</strong> Unspecified</td>
</tr>
<tr>
<td>-cfgName</td>
<td>Configuration name. This name is used to find a matching &lt;converter&gt; tag in the configuration file.</td>
</tr>
<tr>
<td>-D&lt;name&gt;=&lt;value&gt;</td>
<td>Defines a variable that can be used as an alternate replacement for XML environment variables, specified in the form $(VAR_NAME). Note that definitions in the environment take precedence over these definitions.</td>
</tr>
<tr>
<td>-help</td>
<td>Shows this help.</td>
</tr>
</tbody>
</table>
| -verbosity [0-6] | **RTI Service verbosity**  
|                 | 0 - silent         
|                 | 1 - exceptions *(Connext DDS and Converter)*    
|                 | 2 - warnings *(Converter)*    
|                 | 3 - information *(Converter)*    
|                 | 4 - warnings *(Connext DDS and Converter)*    
|                 | 5 - tracing *(Converter)*    
|                 | 6 - tracing *(Connext DDS* and Converter)*    
| **Default:**    | 1 (exceptions) |
| -version         | Prints the program version and exits.                                       |

5.2 Converter Configuration

When you start Converter, you can specify a configuration file in XML format. In this file, you can specify the properties that control the behavior of the service. This section describes how to write a configuration file.

5.2.1 How to Load the XML Configuration

Converter loads its XML configuration from multiple locations. Here are the various sources of configuration files, listed in load order:

1. [working directory]/USER_CONVERTER.xml This file is loaded automatically if it exists.
2. [NDDSHOME]/resource/xml/RTI_CONVERTER.xml This file is loaded automatically if it exists.

**Note:** [working directory] indicates the path to the current working directory from which you run Converter.

[NDDSHOME] indicates the path to your Connext DDS installation. See Section 1.3.

You may use a combination of the above sources and load multiple configuration files.

Here is an example configuration file. You will learn the meaning of each line as you read the rest of this section.
<?xml version="1.0" encoding="UTF-8"?>

<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="https://community.rti.com/schema/6.0.0/rti_converter.xsd">

<!-- Available types -->
<types />

<!-- A definition of a Converter instance to run -->
<converter name="defaultToJson">

<!-- Input storage settings -->
<input_storage>
<sqlite>
<storage_format>XCDR_AUTO</storage_format>
<database_dir>cdr_recording</database_dir>
</sqlite>
</input_storage>

<!-- Output storage settings -->
<output_storage>
<sqlite>
<storage_format>JSON_SQLITE</storage_format>
<fileset>
<workspace_dir>converted</workspace_dir>
<filename_expression>rti_recorder_default.converted.dat</filename_expression>
</fileset>
</sqlite>
</output_storage>

<!-- Domain selection: assume 0 by default. Converter is not a DDS application, so this won’t create a DDS Domain Participant, but allows you to select which domains to convert -->
<domain_participant name="Domain0">
<domain_id>0</domain_id>
</domain_participant>

<session name="DefaultSession" default_participant_ref="Domain0">
<!-- Topics to convert in this session -->
<topic_group name="DefaultTopicGroup">
<!-- Rules describing the topics to convert -->
<allow_topic_name_filter>*</allow_topic_name_filter>
<deny_topic_name_filter>rti/**</deny_topic_name_filter>
</topic_group>
</session>
</converter>
</dds>

5.2. Converter Configuration 95
5.2.2 XML Syntax and Validation

The XML representation of DDS-related resources must follow these syntax rules:

- It shall be a well-formed XML document according to the criteria defined in clause 2.1 of the Extensible Markup Language standard.
- It shall use UTF-8 character encoding for XML elements and values.
- It shall use `<dds>` as the root tag of every document.

To validate the loaded configuration, Converter relies on an XSD file that describes the format of the XML content. We recommend including a reference to this document in the XML file that contains the service’s configuration; this document provides helpful features in code editors such as Visual Studio®, Eclipse®, and NetBeans®, including validation and auto-completion while you are editing the XML file.

The XSD definitions of the XML elements are in `$NDDSHOME/resources/schema/rti_converter.xsd`.

To include a reference to the XSD document in your XML file, use the attribute `xsi:noNamespaceSchemaLocation` in the `<dds>` tag. For example:

```xml
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../schema/rti_converter.xsd">
  <!-- ... -->
</dds>
```

5.2.3 Builtin Configuration of Converter

Converter comes pre-configured with a configuration that converts a file called `rti_recorder_default.dat` from XCDR format to a file called `rti_recorder_default_converted.dat` in JSON_SQLITE format. See Section 5.2.20 for details.

5.2.4 XML Tags for Configuring Converter

This section describes the XML tags you can use in a Converter configuration file. Figure 5.1 and Table 5.2 describe the top-level tags allowed within the root `<dds>` tag.
Table 5.2: Top-Level Tags in Converter’s Configuration file

<table>
<thead>
<tr>
<th>Tags within &lt;dds&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;types&gt;</td>
<td>Defines types that can be used by Converter. This tag is needed if data types are not available in the discovery table. The type description is done using the Connext DDS XML format for type definitions. See Creating User Data Types with Extensible Markup Language (XML), in the RTI Connext DDS Core Libraries User’s Manual.</td>
<td>0..*</td>
</tr>
<tr>
<td>&lt;plugin_library&gt;</td>
<td>Contains a list of storage libraries that you may have implemented to support custom databases. For more information, see Section 3.5.4.</td>
<td>0..*</td>
</tr>
</tbody>
</table>
| <converter>       | **Required.** Specifies a Converter configuration. **Attributes**
|                   | • name: uniquely identifies a Converter configuration. **Required.**
|                   | **Example**
|                   | `<converter name="ConvertAll">`                                           |
|                   | `</converter>`                                                             | 1..*         |

See Section 5.2.5.
5.2.5 Converter Tag

A configuration file must have at least one `<converter>` tag. This tag is used to configure an execution of Converter.

A configuration file may contain multiple `<converter>` tags. When you start Converter, you can specify which `<converter>` tag to use to configure the service using the `-cfgName` command-line parameter. This means one file can be used to configure multiple Converter executions.

Figure 5.2 and Table 5.3 describe the tags allowed within a `<converter>` tag.

![Diagram of Converter Tags in the Converter Configuration File]

Figure 5.2: Converter Tags in the Converter Configuration File
## Table 5.3: Converter Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;converter&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;input_storage&gt;</td>
<td>Describes the storage that Converter will read as input. Converter expects the storage to be in CDR (binary) format. If storage is not specified, data will be stored in a SQLite file using a default name. See Section 5.2.6.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;output_storage&gt;</td>
<td>Describes the storage that Converter will write as output. This output will be in JSON format. If this storage is not specified, data will be stored in a SQLite file using a default name. See Section 5.2.7.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;data_selection&gt;</td>
<td>Enables selection of a subset of data to convert. Supports selecting data for conversion by time (or tagged time). See Section 5.2.14.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| <session>              | Allows you to create one or more sessions in which to convert data, which gives you the option of specifying multiple threads for conversion. See Section 5.2.17. Attributes:  
  - name: Uniquely defines a session. Required.  
  - default_participant_ref: Refers to a domain_participant tag, that specifies the domain ID to use from the database when converting this Topic. Children can override this by specifying their own participant. | 0..* |
| <domain_participant>   | Required. Describes the domains Converter will select for conversion. Converter is not a DDS application so this definition will not create a DDS Domain Participant, it just works as a domain ID selection utility. Note: the domain ID has to correspond to one of the domain IDs defined in the Recorder configuration. See Section 5.2.16. Attributes:  
  - name: Uniquely defines a DomainParticipant. Required. | 1..* |

### Example: Specify a Configuration in XML

```xml
<dds>
  <converter name="ConvertToJSON">
    <!-- ... Required entities -->
  </converter>
</dds>
```

Starting Converter with the following command will use the `<converter>` tag with the name...
“ConvertToJSON”.

```
$NDDSHOME/bin/rticonverter -cfgFile file.xml -cfgName ConvertToJSON
```

5.2.6 Input Storage

The `<input_storage>` tag allows you to configure the storage from which data will be read. You can choose between using the builtin SQLite storage or implementing your own storage plugin.

<table>
<thead>
<tr>
<th>Tags within <code>&lt;input_storage&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;sqlite&gt;</code></td>
<td>Enables converting data from an SQLite database file. See Section 5.2.8.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;legacy&gt;</code></td>
<td>Enables converting data from a legacy version of Recording Service to a database usable by this version. See Section 5.2.11.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;plugin&gt;</code></td>
<td>Enables converting data from storage using an external library that you specify. See Section 5.2.13.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;max_samples_per_read&gt;</code></td>
<td>Specifies the maximum number of samples to be returned by the read() operation. When the recorded tables contain huge amounts of samples, this setting can help improve the responsiveness of Converter, because the tool won’t - for each table - attempt to get all samples at once. If not specified, the default value is 1024 samples. A negative value means an unlimited number of samples.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

5.2.7 Output Storage

The `<output_storage>` tag allows you to configure the storage into which converted data will be written. You can choose between using the builtin SQLite storage or implementing your own storage plugin.

<table>
<thead>
<tr>
<th>Tags within <code>&lt;output_storage&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;sqlite&gt;</code></td>
<td>Enables writing converted data into an SQLite database file. See Section 5.2.8.</td>
<td>0..1</td>
</tr>
<tr>
<td><code>&lt;plugin&gt;</code></td>
<td>Enables writing converted data into storage of your choosing, using a plugin library that you specify. See Section 5.2.13.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

5.2.8 SQLite

The `<sqlite>` tag allows you to specify the name and file extension of a SQLite file to read data from, or write data to. Currently, the only storage format supported for input is CDR, and the
only format supported for output is JSON.

Table 5.6: SQLite Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;sqlite&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;fileset&gt; (output only)</td>
<td>Set of files to write to, and parameters for creating files and directories in that set. Used when creating more than one output file. See Section 5.2.9.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;file&gt; (output only)</td>
<td>Name of file to write converted database file(s) to. Used when creating only a single output file. Default: rti_recorder_default_converted</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;file_suffix&gt; (output only)</td>
<td>Suffix of the output file. Used when creating only a single output file. Default: dat</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;database_dir&gt; (input only)</td>
<td>Directory containing file(s) to convert. Default: Current working directory.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
| <storage_format> | Specifies what format the data is stored in. The options are:  
  - [Input Default] XCDR_AUTO: This is the binary format used by Connext DDS when sending data over the network. This has the highest performance for recording, but can only be viewed by using Converter to convert the data to a readable format, or by using Replay to replay the data. This will internally store data in XCDR or XCDR2 depending on the format received.  
  - JSON_SQLITE: This format can be queried, but recording in this format has lower performance because data must be deserialized before it can be stored.  
  - XCDR: The format to use when communicating with Connext DDS before 6.0.0.  
  - XCDR2: More efficient than XCDR, used by Connext DDS 6.0.0 and later. | 0..1 |
| <overwrite_policy_kind> | Defines what to do when database files are already present in the current Converter execution directory. **There are two options:**  
  - OVERWRITE: delete old files and replace them with newly created ones.  
  - DO_NOT_TOUCH: do not delete any old files and just exit. Default: OVERWRITE | 0..1 |

Continued on next page
Table 5.6 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within <code>&lt;sqlite&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| `<sql_initialization_string>` | Configures a SQL expression to use when establishing SQLite connections using this plugin. You can use this to add an index to a table to speed up replay, but this must be done for each table you want to index. **Example:**

   ```xml
   <sql_initialization_string>
   DROP INDEX IF EXISTS pingtopic_0_by_src_timestamp;
   CREATE INDEX pingtopic_0_by_src_timestamp ON 'PingTopic@0' (SampleInfo_source_timestamp);
   </sql_initialization_string>
   ```
   
   Default: `PRAGMA SYNCHRONOUS = OFF; PRAGMA JOURNAL_MODE = MEMORY;` | 0..1 |

5.2.9 **Fileset**

The `<fileset>` tag allows you to specify a set of files for `Converter` to write to.

The behavior allowed by `Converter` is more limited than the behavior allowed by `Recording Service`.

Table 5.7: Fileset Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within <code>&lt;fileset&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;workspace_dir&gt;</code></td>
<td><strong>Required.</strong> Base directory where <code>Converter</code> should output files. Must be different from the input <code>database_dir</code> directory. Default: The current working directory</td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 5.7 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within <code>&lt;file-set&gt;</code></th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;filename_expression&gt;</code></td>
<td><strong>Required.</strong> The file name <em>Converter</em> will use for the generated user data file(s). This expression in combination with the rollover configuration will determine how many (and the names of) files <em>Converter</em> will output. This setting accepts text and any combination of the following parameters in it:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Autonumeric. <strong>Format:</strong> <code>%auto:M%</code>. This parameter describes an integer that auto-increments every time <em>Converter</em> has a rollover event due to time or file size.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- <strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;filename_expression&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>test_files_%auto:0%.dat</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;filename_expression&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This will create a series of files starting with test_files_0.dat, and incrementing each time <em>Converter</em> rolls over (due to the file size or time limits being reached).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Timestamp. <strong>Format:</strong> <code>%ts%</code>. This parameter will take the current timestamp in the system (the time represented as number of seconds since Epoch).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Time. <strong>Format:</strong> <code>%T%</code>. Current time expressed in ISO 8601 time format (HH:MM:SS). Example: 14:55:02. This parameter uses the strftime() parameter <code>%T</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Short date. <strong>Format:</strong> <code>%F%</code>. Short date in YYYY-MM-DD format. Example: 2001-08-23. This parameter uses the strftime() parameter <code>%F</code>. <strong>Note:</strong> this parameter will not vary in 24 hours, so use with caution in combination with the rollover time limit feature (time limit should be greater than 1 day; otherwise, you may overwrite the same file continuously).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Date and time. <strong>Format:</strong> <code>%c%</code>. Date and time representation, locale-dependent. This parameter uses the strftime() parameter <code>%c</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: <code>rti_recorder_default_%auto:0%.dat</code> (auto-numeric starting at zero, unlimited)</td>
<td></td>
</tr>
<tr>
<td><code>&lt;rollover&gt;</code></td>
<td>Configuration for rolling over the file after a size limit is reached. See Section 5.2.10.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
### 5.2.11 Legacy

This tag configures how legacy databases are converted to the newer format. (Note: You can continue to use the older format with *Replay*, by specifying “legacy” storage in *Replay’s* configuration.)

<table>
<thead>
<tr>
<th>Tags within &lt;legacy&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;file_path&gt;</td>
<td>The path to a legacy file to be converted. File set and version properties will be obtained automatically from the file itself.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;domain_mapping&gt;</td>
<td>This tag allows you to link legacy domain names with domain IDs. See Section 5.2.12.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

### 5.2.12 Domain Mapping

When converting a legacy database, the domain may not have been recorded. This tag provides a way to map a table with a domain ID.

Later versions of the old Recorder database allowed you to use field filters. By default, the domain ID field was not recorded (it was filtered out by default). Thus, there is no information available in these legacy databases to relate the domain name used to record the data with a domain ID. This tag allows you to link legacy domain names with domain IDs.

<table>
<thead>
<tr>
<th>Tags within &lt;domain_mapping&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <domain_map>                | **Required.** A link between a recorded legacy domain name and a domain ID. **Attributes:**  
  - `legacy_domain_name`: The recorded domain name specified in the old Recording Service domain tag, for example: `<domain name="domain0">`.  
  - `domain_id`: The domain ID you want to associate with the legacy domain name. | 1..* |
5.2.13 Plugin

This tag enables you to convert data using an external library that you specify.

Table 5.11: Plugin Tags in the Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;plugin&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;property&gt;</td>
<td>Name/value pairs of properties to pass to a storage plugin.</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

5.2.14 Data Selection

This tag selects what data to convert from the database, based on a time range.

Table 5.12: Data Selection Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;data_selection&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;time_range&gt;</td>
<td>Select data to convert from the database, based on start and end times. See Section 5.2.15.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

5.2.15 Time Range

The <time_range> tag allows you to specify the begin and end times of the data you want to convert. This can be specified either as timestamps or as symbolic timestamps called “timestamp tags.” Tags may have been added to the recording through remote administration, and can be viewed using the script `rtirecordingservice_list_tags` (see Section 3.5.7).

Table 5.13: Time Range Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;time_range&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;begin_time&gt;</td>
<td>Select data to convert from the database, with a timestamp beginning at this time. Specify the time in seconds and nanoseconds. Default: 0, start at beginning of the file.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;begin_tag&gt;</td>
<td>Can be used instead of begin_time. Select data to start converting from the database by specifying the string name of a timestamp tag. Timestamp tags associate a timestamp with a name. Then you can refer to a timestamp by that name. The timestamp must have been tagged with that name during recording. See Section 3.5.6. Default: Start at beginning of the file.</td>
<td>0..1</td>
</tr>
<tr>
<td>&lt;end_time&gt;</td>
<td>Select data to convert from the database, with a timestamp ending at this time. Specify the time in seconds and nanoseconds. Default: End at end of file.</td>
<td>0..1</td>
</tr>
</tbody>
</table>
Table 5.13 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;time_range&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;end_tag&gt;</td>
<td>Can be used instead of end_time. Select when to stop converting data by specifying the string name of a timestamp tag. Timestamp tags associate a timestamp with a name. Then you can refer to a timestamp by that name. The timestamp must have been tagged with that name during recording. See Section 3.5.6. Default: Stop at end of file.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

5.2.16 DomainParticipant

The <domain_participant> tag allows you to specify the domain IDs you want to convert from the database. Specify a name for the DomainParticipant, and the domain_id you want to convert. Associate the DomainParticipant name with a session, topic or topic_group that you want to convert, to convert data within a domain.

This does not create a DDS DomainParticipant.

Table 5.14: DomainParticipant Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;domain_participant&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;domain_id&gt;</td>
<td>The domain ID of tables loaded from the database. This is used to determine which domain IDs to convert.</td>
<td>1</td>
</tr>
<tr>
<td>&lt;register_type&gt;</td>
<td>Registers a type name and associates it with a type representation. When you define a type in the configuration file, you have to register the type in order to use it in a &lt;topic&gt;.</td>
<td>0..*</td>
</tr>
</tbody>
</table>

5.2.17 Session

The <session> tag configures the threads that will be used to convert data. You also specify the Topics and groups of Topics to convert inside the <session> tag.

Table 5.15: Session Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;session&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;thread_pool&gt;</td>
<td>Specifies the number of threads used by this session, and the settings used when creating them.</td>
<td>0..1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 5.15 – continued from previous page

<table>
<thead>
<tr>
<th>Tags within &lt;session&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <topic>               | Specifies an individual Topic to convert. See Section 5.2.19. **Attributes:**  
  - name: The name of the Topic to convert.  
  - participant_ref: Refers to a domain_participant tag, that specifies the domain ID to use from the database when converting this Topic. If the parent <session> specifies a default_participant_ref, this attribute is optional. | 0..* |
| <topic_group>         | Specifies a group of Topics to convert. See Section 5.2.18. **Attributes:**  
  - name: The name of the topic group.  
  - participant_ref: Refers to a domain_participant tag, that specifies the domain ID to use from the database when converting this Topic. If the parent <session> specifies a default_participant_ref, this attribute is optional. | 0..* |

5.2.18 Topic Group

The <topic_group> tag allows you to convert a group of Topics, using regular expressions to describe which Topics to convert.

Table 5.16: Topic Group Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;topic_group&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
</table>
| <allow_topic_name_filter> | A regular expression describing which Topics should be converted. **Example:**  
  ```xml  
  <topic_group name="ConvertAll">  
  <allow_topic_name_filter>*</allow_topic_name_filter>  
  </topic_group>  
  ``` | 0..1 |
| <deny_topic_name_filter> | A regular expression describing which Topics should not be converted. This is applied after the allow_topic_name_filter. | 0..1 |

5.2.19 Topic

The <topic> tag specifies an individual Topic to convert.
Table 5.17: Topic Tags in Converter’s Configuration File

<table>
<thead>
<tr>
<th>Tags within &lt;topic&gt;</th>
<th>Description</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;registered_type_name&gt;</td>
<td>The name of the data type that will be converted for this topic. <strong>Required.</strong></td>
<td>1</td>
</tr>
</tbody>
</table>
| <transformation>           | The transformation library to be applied to this Topic’s data when converting. This is a user library that can modify the data after it is received from input storage and before it is sent to output storage. Transformations implement APIs identical to Routing Service’s transformations. For more on using transformations, see these sections in the RTI Routing Service User’s Manual:  
  • Data Transformation  
  • Tutorials  
Attributes:  
  • plugin_name: The name of the plugin to load, qualified by the plugin library name. | 0..1          |

Example:

```xml
<dds>
  <plugin_library name="ConverterTransformations">
    <transformation_plugin name="ModifyTestID">
      <create_function>ModifyTestID_create</create_function>
      <dll>modify_test_id_library</dll>
    </transformation_plugin>
  </plugin_library>
  <!-- ... -->
  <converter>
    <!-- ... -->
    <topic name="TestTopic">
      <transformation plugin_name="ConverterTransformations::ModifyTestID" />
    </topic>
  </converter>
</dds>
```

5.2.20 Converter’s Builtin Configuration Details

Converter comes with a builtin configuration, which selects the name of an input file to convert from and an output file to convert to. The ‘defaultToJson’ configuration specifies JSON SQLite format by default as the output format, with ‘rti_recorder_default.dat’ as the input file and ‘rti_recorder_default_converted.dat’ as the output file. It expects to find the file(s) to convert in a directory called ‘cdr_recording’. This is the default directory when recording in XCDR format in the default Recorder configuration.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- ... -->
  <converter>
    <!-- ... -->
    <topic name="TestTopic">
      <transformation plugin_name="ConverterTransformations::ModifyTestID" />
    </topic>
  </converter>
</dds>
```
5.3 Tutorials

5.3.1 Using Timestamp Tags with Converter

If your recording was originally made with the builtin SQLite storage plugin, and you used the tag_timestamp remote command to tag certain events, then your recording contains timestamp tags: symbolic timestamp names you can use in place of timestamps expressed in units of time. For more information on timestamp tags, see Section 3.5.6.

You can list the timestamp tags that are in your recorded database by using the rtirecordingservice_list_tags script. Use the -d argument to point to the directory that contains your recorded database, as follows:

```
<NDDSHOME>/bin/rtirecordingservice_list_tags -d /database/directory/
```
This command will analyze the recording in /database/directory/ and list the details of the timestamp tags in the recording, including the tag names, their descriptions, and associated timestamps.

You can use the tag_name of the tags you find in a recording when you are creating an XML configuration file for Converter by using a <data_selection> tag.

For example, if after running `rtirecordingservice_list_tags`, you see output such as:

<table>
<thead>
<tr>
<th>tag_name</th>
<th>timestamp_ms</th>
<th>tag_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/my_example/my_events/tag1</td>
<td>1546484663309</td>
<td>first tag description</td>
</tr>
<tr>
<td>/my_example/my_events/tag2</td>
<td>1546484703360</td>
<td>a second tag description</td>
</tr>
</tbody>
</table>

Then you can have a <data_selection> tag in your XML for Converter, after the <output_storage> tag, that looks like the following:

```xml
<data_selection>
  <time_range>
    <begin_tag>/my_example/my_events/tag1</begin_tag>
    <end_tag>/my_example/my_events/tag2</end_tag>
  </time_range>
</data_selection>
```

Converter will convert data between those tags. Note that when expressing a <time_range> tag, you can mix and match timestamps and tags. For example, you can use a <begin_tag> (by referring to a tag_name) to express the time when conversion should begin, and use an <end_tag> with an end_time timestamp (expressed in time units) to specify when conversion should end. If you do not provide one of the bounds, the start of recording is the default begin bound, and the end of recording is the default end bound.
Chapter 6

Common Infrastructure

6.1 Application Resource Model

RTI services are described through a *hierarchical application resource model*. In this model, an application is composed of a set of *Resources*, each representing a particular component within the application. *Resources* have a parent-child relationship. Figure 6.1 shows a general view of this concept.

![Diagram of Application modeled as a set of related Resources](image)

**Figure 6.1:** Application modeled as a set of related Resources

Each application specifies its resource model by indicating the available resources and their relationship. A *Resource* is determined by its class and a concrete object instance. It can belong to one of the following categories:

- **Simple** – Represents a single object.
• **Collection**—Represents a set of objects of the same class.

A Resource may be composed of one or more Resources. In this relationship, the *parent* Resource is composed of one or more *child* Resources.

**6.1.1 Example: Simple Resource Model of a Connext DDS Application**

Figure 6.2 depicts a UML class diagram to provide a generic resource model for *Connext DDS* applications.

![UML class diagram](image)

Figure 6.2: Connext DDS application resource model

In this diagram, the composition relationship is used to denote the parents and children in the hierarchy. The direct relationship denotes a dependency between resources that is not parent-child.

**Resource Identifiers**

A resource identifier is a string of characters that uniquely address a concrete resource object within an application. It is expressed as a hierarchical sequence of identifiers separated by /, including all the parent resources and the target resource itself:

```
/resource_id1/resource_id2.../resource_idN
```

where each individual identifier references a concrete resource object *by its name*. The object name is either:

1. Fixed and specified by the resource model of the parent Resource class.
2. Given by the user of the application. This is the case where the parent resource is a collection in which the user can insert objects, providing a name for each of them.

The individual identifier can refer to one of the two kinds of resources, simple and collection resources. For example:
If the identifier refers to a collection resource, the following child identifier must refer to a simple resource. Both simple and collection resources can be parents (or children). In the previous example, resource_id_1 is a simple resource child of collection_id_1; it is also the parent of resource_id_2.

The hierarchy of identifiers is known as the full resource identifier path, where each resource on the left represents a parent resource. The full resource identifier path is composed of collection and simple resources. Each child resource identifier is known as the relative resource to the parent.

The resource identifier format follows these conventions:

- The first character is /, which represents the root resource and parent of all the available resources across the applications.
- A collection identifier is defined in lower snake_case, and it is always specified by the resource class.
- A simple resource identifier is defined in camelCase (lower and upper) and may be specified by both the resource class or the user.

**6.1.2 Example: Resource Identifiers of a Generic Connext DDS Application**

Consider the Connext DDS application resource model in Section 6.1.1. The following resource identifier addresses a concrete DomainParticipant named “MyParticipant” in a given application:

```
/demand_participants/MyParticipant
```

In this case, “domain_participants” is the identifier of a collection resource that represents a set of DomainParticipants in the application and its value is fixed and specified by the application. In contrast, “MyParticipant” is the identifier of a simple resource that represents a particular DomainParticipant and its value is given by the user of the application at DomainParticipant creation time.

The following resource identifier addresses the implicit Publisher of a concrete DomainParticipant in a given application:

```
/demand_participants/MyParticipant/implicit_publisher
```

where “implicit_publisher” is the identifier of a simple resource that represents the always-present implicit Publisher and its value is fixed and specified by the DomainParticipant resource class.

**6.2 Remote Administration Platform**

This section describes details of the RTI Remote Administration Platform, which represents the foundation of the remote access capabilities available in RTI Routing Service, RTI Recording Service, and RTI Queuing Service. The RTI Remote Administration Platform provides a common infrastructure that unifies and consolidates the remote interface to all RTI services.
Note: Remote administration of RTI services requires an understanding of the application resource model. We recommend that you read Application Resource Model (Section 6.1) before continuing with this section.

The RTI Remote Administration Platform addresses two areas:

- **Resource Interface**: How to perform operations on a set of resource objects that are available as part of the public interface of the remote service.

- **Communication**: How the remote service receives and sends information.

The combination of these two areas provides the general view of the RTI Remote Administration Platform, as shown in Figure 6.3. The RTI Remote Administration Platform is defined as a request/reply architecture. In this architecture, the service is modeled as a set of resources upon which the requester client can perform operations. Resources represent objects that have both state and behavior.

![Figure 6.3: General View of the RTI Remote Administration Platform Architecture](image)

Clients issue requests indicating the desired operation and receive replies from the service with the result of the requests. If multiple clients issue multiple requests to one or more services, the client will receive only replies to its own requests.

### 6.2.1 Remote Interface

Services offer their available functionality through their set of resources. The RTI Remote Administration Platform defines a Representational State Transfer (REST)-like interface to address service resources and perform operations on them. A resource operation is determined by a REST request and the associated result by a REST reply.
Table 6.1: REST Interface

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST Request</td>
<td>[method] + [resource_identifier] + [body]</td>
</tr>
<tr>
<td></td>
<td>• method: Specifies the action to be performed on a service resource. There is only a small subset of methods, known as standard methods (see Section 6.2.1).</td>
</tr>
<tr>
<td></td>
<td>• resource_identifier: Addresses a concrete service resource. Each concrete service has its own set of resources (see Section 6.1.1).</td>
</tr>
<tr>
<td></td>
<td>• body: Optional request data that contains necessary information to complete the operation.</td>
</tr>
<tr>
<td>REST Reply</td>
<td>[return code] + [body]</td>
</tr>
<tr>
<td></td>
<td>• return code: Integer indicating the result of the operation.</td>
</tr>
<tr>
<td></td>
<td>• body: Optional reply data that contains information associated with the processing of the request.</td>
</tr>
</tbody>
</table>

Standard Methods

The RTI Remote Administration Platform defines the methods listed in Table 6.2.

Table 6.2: Standard Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
<th>Request Body</th>
<th>Reply Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Parent collection resource identifier</td>
<td>Resource representation</td>
<td>N/A</td>
</tr>
<tr>
<td>GET</td>
<td>Resource identifier</td>
<td>N/A</td>
<td>Resource representation</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Resource identifier</td>
<td>Resource representation</td>
<td>N/A</td>
</tr>
<tr>
<td>DELETE</td>
<td>Resource identifier</td>
<td>Undefined</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Custom Methods

There are certain cases in which an operation on a service resource cannot be mapped intuitively to a standard method and resource identifier. Custom methods address this limitation.

A custom method can be specified as part of the resource identifier, after the resource path, separated by a ::

```
UPDATE + [resource_identifier] : [custom_verb]
```

It is up to each service implementation to define which custom methods are available and on what
resources they apply. Custom methods follow these conventions:

- They are invoked through the **UPDATE** standard method.
- They are named using lower snake_case.
- They may use the request body and reply body if necessary.

**Example: Database Rollover**

This example shows the REST request to perform a file rollover operation on a file-based database:

```
UPDATE /databases/MyDatabase:rollover
```

### 6.2.2 Communication

The information exchange between client and server is based on the DDS request-reply pattern, as shown in Figure 6.4. The client maps to a **Requester**, whereas the server maps to a **Replier**.

![Communication in RTI Remote Administration Platform](image)

**Figure 6.4:** Communication in *RTI Remote Administration Platform* is Based on DDS Request-Reply

The communication is performed over a single request-reply channel, composed of two topics:

- **Command Request Topic:** Topic through which the client sends the requests to the server.
- **Command Reply Topic:** Topic through which the server sends the replies to the received requests.

The definition of these topics is shown in Table 6.3:
Table 6.3: Remote Administration Topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Name</th>
<th>Top-level Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommandRequestTopic</td>
<td>rti/service/administration/command_request</td>
<td>rti::service::admin::CommandRequest</td>
</tr>
<tr>
<td>CommandReplyTopic</td>
<td>rti/service/administration/command_reply</td>
<td>rti::service::admin::CommandReply</td>
</tr>
</tbody>
</table>

The definition for each Topic type is described below.

Listing 6.1: CommandRequest Type

```c
@appendable
class CommandRequest {
    @key int32 instance_id;
    @optional string<BOUNDED_STRING_LENGTH_MAX> application_name;
    CommandActionKind action;
    ResourceIdentifier resource_identifier;
    StringBody string_body;
    OctetBody octet_body;
};
```

Table 6.4: CommandRequest

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance_id</td>
<td>Associates a request with a given instance in the CommandRequestTopic. This can be used if your requester application model wants to leverage outstanding requests. In general, this member is always set to zero, so all requests belong to the same CommandRequestTopic instance.</td>
</tr>
<tr>
<td>application_name</td>
<td>Optional member that indicates the target service instance where the request is sent. If NULL, the request will be sent to all services.</td>
</tr>
<tr>
<td>action</td>
<td>Indicates the resource operation.</td>
</tr>
<tr>
<td>resource_identifier</td>
<td>Addresses a service resource.</td>
</tr>
<tr>
<td>string_body</td>
<td>Contains content represented as a string.</td>
</tr>
<tr>
<td>octet_body</td>
<td>Contains content represented as binary.</td>
</tr>
</tbody>
</table>
@appendable
struct CommandReply {
    CommandReplyRetcode retcode;
    int32 native_retcode;
    StringBody string_body;
    OctetBody octet_body;
};

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retcode</td>
<td>Indicates the result of the operation.</td>
</tr>
<tr>
<td>native_retcode</td>
<td>Provides extra information about the result of the operation.</td>
</tr>
<tr>
<td>string_body</td>
<td>Return value of the operation, represented as a string.</td>
</tr>
<tr>
<td>octet_body</td>
<td>Return value of the operation, represented as binary.</td>
</tr>
</tbody>
</table>

The type definitions for both the `CommandRequestTopic` and `CommandReplyTopic` are in the file `[NDDSHOME]/resource/idl/ServiceAdmin.idl`.

The definition of the request and reply topics is independent of any specific service implementation. In fact, the topic names are fixed, unique, and shared across all services that rely on the *RTI Remote Administration Platform*. Clients can target specific services through two mechanisms:

- Specifying a concrete service instance by providing its *application name*. The application name is a service attribute and can be set at service creation time.
- Specifying the configuration name loaded by the target services. The target service configuration shall be present in the service resource part of the `resource_identifier`.

**Reply Sequence**

Usually a request is expected to generate a single reply. Sometimes, however, a request may trigger the generation of multiple replies, all associated with the same request.

The *RTI Remote Administration Platform* communication architecture allows services to respond to certain requests with a *reply sequence*. All the samples in a reply sequence use the metadata `SampleFlagBits` to indicate whether it belongs to a reply sequence and whether there are more replies pending.

The `SampleFlagBits` may contain different flags that indicate the status of the reply procedure. For a given reply sequence, the associated sample flags for each reply may contain:

- `SEQUENTIAL_REPLY`: If present, this indicates that the sample is the first reply of a reply sequence and there are more on the way.
- `FINAL_REPLY`: If present, this indicates that the sample is the last one belonging to a reply sequence. This flag is valid only if the `SEQUENTIAL_REPLY` is also set.
For more on SampleFlagBits, see documentation on the DDS_SampleInfo structure in the Connext DDS API Reference HTML documentation.

**Example: Accessing from Connext DDS Application**

This example shows a Modern C++ snippet on how to use Connext DDS Request-Reply to disable a Routing Service instance.

```cpp
using namespace rti::request;
using namespace dds::core;
using namespace RTI::Service;
using namespace RTI::Service::Admin;

const unsigned int WAIT_TIMEOUT_SEC_MAX = 10;
const unsigned int ADMIN_DOMAIN_ID = 55;

int main(int argc, char *argv[]) {
    try {
        dds::domain::DomainParticipant participant(ADMIN_DOMAIN_ID);

        // create requester params
        rti::request::RequesterParams requester_params(participant);
        requester_params.request_topic_name(COMMAND_REQUEST_TOPIC_NAME);
        requester_params.reply_topic_name(COMMAND_REPLY_TOPIC_NAME);

        // Wait for Routing Service Discovery
        dds::core::status::PublicationMatchedStatus matched_status;
        unsigned int wait_count = 0;

        std::cout << "Waiting for a matching replier..." << std::endl;
        int wait_count = 0;
        while (matched_status.current_count() < 1
               && wait_count < WAIT_TIMEOUT_SEC_MAX) {
            matched_status = requester.request_datawriter().publication_matched_status();
            wait_count++;
            rti::util::sleep(Duration(1));
        }

        if (matched_status.current_count() < 1) {
            throw dds::core::Error("No matching replier found.");
        }

        /* * Setup command */
        CommandRequest request;
        request.action(CommandActionKind::RTI_SERVICE_COMMAND_ACTION_UPDATE);
        request.resource_identifier("/routing_services/MyRouter/state");
        dds::topic::topic_type_support<EntityState>::to_cdr_buffer(
            static_cast<std::vector<char> &>(request.octet_body()),
            EntityState(RTI_SERVICE_DISABLED_ENTITY_STATE));
    }
}
```

(continues on next page)
6.2.3 Common Operations

The set of services that use the RTI Remote Administration Platform to implement remote administration also share a base remote interface that consolidates and unifies the semantics and behavior of certain common operations.

Services containing resources that implement the common operations conform to the base remote interface, making sure that signatures, semantics, behavior, and conditions are respected.

The following sections describe each of these common operations.

Create Resource

CREATE [resource_identifier]

Creates a resource object from its configuration in XML representation.

This operation creates a resource object and its contained entities. The created object becomes a child of the parent specified in the resource_identifier.

After successful creation, the resource object is fully addressable for additional remote access, and the associated object configuration is inserted into the currently loaded full XML configuration.

Request body

- string_body: XML representation of the resource object provided as file:// or str://.

Example str:// request body:
str::"<my_resource name="NewResourceObject">
...
</my_resource>"

- Example file:// request body:

  file://home/rti/config/service_my_resource.xml

Reply body
- Empty.

Return codes
The operation may return a reply with error if:
- The specified resource identifier does not exist.
- The specified configuration is schematically invalid.
- There was an error creating the resource object.

Get Resource
GET [resource_identifier]
Returns an equivalent XML string that represents the current state of the resource object configuration, including any updates performed during its lifecycle.

Request body
- Empty.

Reply body
- string_body: XML representation of the resource object.
- Example reply body:

  <my_resource name="MyObject">
  ...
  </my_resource>

Return codes
The operation may return a reply with error if:
- The specified resource identifier does not exist.

Update Resource
UPDATE [resource_identifier]
Updates the specified resource object from its configuration in XML representation.

This operation modifies the properties of the resource object, including the associated configuration. Only the mutable properties of the resource class can be updated while the object is enabled. To update immutable properties, the resource object must be disabled first.
Implementations may validate the receive configuration against a scheme (DTD or XSD) that defines the valid set of accepted parameters (for example, only mutable elements).

**Request body**

- **string_body**: XML representation of the resource object provided as `file://` or `str://`.
- Example `str://` request body:

```xml
str://"<my_resource name="MyResourceObject">...
</my_resource>"
```

- Example `file://` request body:

```xml
file:///home/rti/config/service_update_my_resource.xml
```

**Reply body**

- Empty.

**Return codes**

The operation may return a reply with error if:

- The specified resource identifier does not exist.
- The specified configuration is schematically invalid.
- The specified configuration contains changes in immutable properties.
- There was an error updating the resource object.

**Set Resource State**

**UPDATE [resource_identifier]/state**

Sends a state change request to the specified resource object.

This operation attempts to change the state of the specified resource object and propagates the request to the resource object’s contained entities.

The target state must be one of the resource class’s valid accepted states.

**Request body**

- **octet_body**: CDR representation of an entity state.

**Reply body**

- Empty.

**Return codes**

The operation may return a reply with error if:

- The specified resource identifier does not exist.
- The target request is invalid.
• The resource object reported an error while performing the state transition.

Delete Resource

DELETE [resource_identifier]

Deletes the specified resource object.

This operation deletes a resource object and its contained entities. The deleted object is removed from its parent resource object.

The associated object configuration is removed from the currently loaded full XML configuration.

After a successful deletion, the resource object is no longer addressable for additional remote access.

Request body
• Empty.

Reply body
• Empty.

Return codes

The operation may return a reply with error if:
• The specified resource identifier does not exist.
• There was an error deleting the resource object.

6.3 Monitoring Distribution Platform

Monitoring refers to the distribution of health status information metrics from instrumented RTI services. This section describes the architecture of the monitoring capability supported in RTI Routing Service and RTI Recording Service. You will learn what type of information these applications can provide and how to access it.

RTI services provide monitoring information through a Distribution Topic, which is a DDS Topic responsible for distributing information with certain characteristics about the service resources. An RTI service provides monitoring information through the following three distribution topics:

• ConfigDistributionTopic: Distributes metrics related to the description and configuration of a Resource. This information may be immutable or change rarely.

• EventDistributionTopic: Distributes metrics related to Resource status notifications of asynchronous nature. This information is provided asynchronously when Resources change after the occurrence of an event.

• PeriodicDistributionTopic: Distribute metrics related to periodic, sampling-based updates of a Resource. Information is provided periodically at a configurable publication period.

These three Topics are shared across all services for the distribution of the monitoring information. Table 6.6 provides a summary of these topics.
Table 6.6: Monitoring Distribution Topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Name</th>
<th>Top-level Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigDistributionTopic</td>
<td>rti/service/monitoring/config</td>
<td>rti::service::monitoring::Config</td>
</tr>
<tr>
<td>EventDistributionTopic</td>
<td>rti/service/monitoring/event</td>
<td>rti::service::monitoring::Event</td>
</tr>
<tr>
<td>PeriodicDistributionTopic</td>
<td>rti/service/monitoring/periodic</td>
<td>rti::service::monitoring::Periodic</td>
</tr>
</tbody>
</table>

Figure 6.5 shows the mapping of the monitoring information into the distribution Topics. A distribution Topic is keyed on service resources categorized as keyed Resources. These are resources whose related monitoring information is provided as an instance on the distribution Topic.

6.3.1 Distribution Topic Definition

All distribution Topics have a common type structure that is composed of two parts: a base type that identifies a resource object and a resource-specific type that contains actual status monitoring information.

The definition of a distribution Topic is shown in Figure 6.6.

Keyed Resource Base Type Fields

This is the base type of all distribution Topics and consists of two fields:

- **object_guid**: Key field. It represents a 16-byte sequence that uniquely identifies a Keyed Resource across all the available services in the monitoring domain. Hence, the associated instance handle key hash will be the same for all distribution Topics, allowing easy correlation.
RTI Recording Service Documentation, Version 6.0.0

Figure 6.6: Monitoring Distribution Topic Definition

of a resource. It will also facilitate, as we will discuss later, easy instance data manipulation in a DataReader.

- **parent_guid**: It contains the object GUID of the parent resource. This field will be set to all zeros if the object is a top-level resource thus with no parent.

This base type, KeyedResource, is defined in [NDDSHOME]/resource/idl/ServiceCommon.idl.

**Resource-Specific Type Fields**

This is the type that conveys monitoring information for a concrete resource object. Since a distribution Topic is responsible for providing information about different resource classes, the resource-specific type consists of a single field that is a **Union of all the possible representations** for the keyed resources that provide that on the topic.

As expected, there must be consistency between the two parts of the distribution topic type. That is, a sample for a concrete resource object must contain the resource-specific union discriminator corresponding to the resource object’s class.

**Example: Monitoring of Generic Application**

Assume a generic application that provides monitoring information about the modes of transports Car, Boat and Plane. Each mode is mapped to a keyed resource, each with a custom type that contains metrics specific to each class.

The monitoring distribution Topic top-level type, TransportModeDistribution, would be defined as follows, using IDL v4 notation:

```idl
#include "ServiceCommon.idl"

@nested
struct CarType {
    float speed;
    String color;
    String plate_number;
}
```
Assume now that in the monitoring domain there are three resource objects, one for each resource class: a `Car` object ‘CarA’, a `Boat` object ‘Boat1’, and a `Plane` object ‘PlaneX’. They all have unique resource GUIDs and each object represents an instance in the distribution `Topic`. The table shows the example of potential sample values:

<table>
<thead>
<tr>
<th>object_guid</th>
<th>CarA</th>
<th>Boat1</th>
<th>PlaneX</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent_guid</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>value discriminator</td>
<td>CAR_TRANSPORT_MODE</td>
<td>BOAT_TRANSPORT_MODE</td>
<td>PLANE_TRANSPORT_MODE</td>
</tr>
</tbody>
</table>

### Table 6.7: Samples in TransportModeDistribution `Topic`

6.3. Monitoring Distribution Platform
6.3.2 DDS Entities

RTI services allow you to distribute monitoring information in any domain. For that, they create the following DDS entities:

- A DomainParticipant on the monitoring domain.
- A single Publisher for all DataWriters.
- A DataWriter for each distribution Topic.

A service will create these entities with default QoS or otherwise the corresponding service user’s manual will specify the actual values. Services allow you to customize the QoS of the DDS entities, typically in the service monitoring configuration under the <monitoring> tag. You will need to refer to each service’s user’s manual.

6.3.3 Monitoring Metrics Publication

How services publish monitoring samples depends on the distribution Topic.

Configuration Distribution Topic

There are two events that cause the publication of samples in this topic:

- As soon as a Resource object is created. This event generates the first sample in the Topic for the resource object just created. Since these first samples are published as resources are created, it is guaranteed to be in hierarchical order; that is, the sample for a parent Resource is published before its children. When Resources are created depends on the service. Typically, Resources are created on service startup. Other cases include manual creation (e.g., through remote administration) or external event-driven creation (e.g., discovery of matching streams, in the case of AutoRoute in Routing Service).

- On Resource object update. This event occurs when the properties of the object change due to a set or update operation (e.g., through remote administration).

Event Distribution Topic

Services publish samples in this Topic in reaction to an internal event, such as a Resource state change. Which events and their associated information and when they occur is highly dependent on concrete service implementations.

Periodic Distribution Topic

Samples in this Topic are published periodically, according to a fixed configurable period. The metrics provided in this Topic are generated in two different ways:

- As a snapshot of the current value, taken at the publication time (e.g., current number of matching DataReaders). This represents a simple case and the metric is typically represented with an adequate primitive member.

- As a statistic variable generated from a set of discreet measurements, obtained periodically. This represents a continuous flow of metrics, represented with the StatisticVariable type (see Section 6.3.4).
There are two activities involved in the generation of the statistic variables: Calculation and Publication. All the configuration elements for these activities are available under the `<monitoring>` tag.

**Calculation**

The instrumented service periodically performs measurements on the metric. This activity is also known as *sampling* (don't confuse with data samples). The frequency of the measurements can be configured with the tag `<statistics_sampling_period>`. As a general recommendation, the sampling period should be a few times smaller than the publication period. A small sampling period provides more accurate statistics generation at the expense of increasing memory and CPU consumption.

**Publication**

The service periodically publishes a data sample containing a snapshot of the statistics generated during the calculation phase. The publication period can be configured with the tag `<statistics_publication_period>`. The value of a statistic variable corresponds to the time window of a publication period.

**6.3.4 Monitoring Metrics Reference**

This section describes the types used as common metrics across services. All the type definitions listed here are in `[NDDSHOME]/resource/idl/ServiceCommon.idl`. 

**Statistic Variable**
Listing 6.3: Statistics

```c
@appendable @nested
struct StatisticMetrics {
    int64 period_ms;
    int64 count;
    float mean;
    float minimum;
    float maximum;
    float std_dev;
};

@appendable @nested
struct StatisticVariable {
    StatisticMetrics publication_period_metrics;
};
```

Table 6.8: StatisticMetrics

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>period_ms</td>
<td>Period in milliseconds at which the metrics are published.</td>
</tr>
<tr>
<td>count</td>
<td>Sum of all the measurement values obtained during the publication period.</td>
</tr>
<tr>
<td>mean</td>
<td>Arithmetic mean of all the measurement values during publication period. For aggregated metrics, this value is the mean of all the aggregated metrics means.</td>
</tr>
<tr>
<td>min</td>
<td>Minimum of all the measurement values during publication period. For aggregated metrics, this value is the minimum of all the aggregated metrics minimums.</td>
</tr>
<tr>
<td>max</td>
<td>Maximum of all the measurement values during publication period. For aggregated metrics, this value is the maximum of all the aggregated metrics minimums.</td>
</tr>
<tr>
<td>std_dev</td>
<td>Standard deviation of all the measurement values during publication period. For aggregated metrics, this value is the standard deviation of all the aggregated metrics minimums.</td>
</tr>
</tbody>
</table>

Host Metrics

Listing 6.4: Host Types

```c
@appendable @nested
struct HostPeriodic {
    @optional StatisticVariable cpu_usage_percentage;
    @optional StatisticVariable free_memory_kb;
    @optional StatisticVariable free_swap_memory_kb;
    int32 uptime_sec;
};
```

(continues on next page)
struct HostConfig {
    BoundedString name;
    uint32 id;
    int64 total_memory_kb;
    int64 total_swap_memory_kb;
};

Table 6.9: HostConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the host where the service is running.</td>
</tr>
<tr>
<td>id</td>
<td>ID of the host where the service is running.</td>
</tr>
<tr>
<td>total_memory_kb</td>
<td>Total memory in KiloBytes of the host where the service is running. Ability of this value is platform dependent.</td>
</tr>
<tr>
<td>total_swap_memory_kb</td>
<td>Total swap memory in KiloBytes of the host where the service is running. Availability of this value is platform dependent.</td>
</tr>
</tbody>
</table>

Table 6.10: HostPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_usage_percentage</td>
<td>Statistic variable that provides the global percentage of CPU usage on the host where the service is running. Availability of this value is platform dependent.</td>
</tr>
<tr>
<td>free_memory_kb</td>
<td>Statistic variable that provides the amount of free memory in KiloBytes of the host where the service is running. Availability of this value is platform dependent.</td>
</tr>
<tr>
<td>free_swap_memory_kb</td>
<td>Statistic variable that provides the amount of free swap memory in KiloBytes of the host where the service is running. Availability of this value is platform dependent.</td>
</tr>
<tr>
<td>uptime_sec</td>
<td>Time in seconds elapsed since the host on which the running service started. Availability of this value is platform dependent.</td>
</tr>
</tbody>
</table>

Process Metrics

Listing 6.5: Process Types

@appendable @nested
struct ProcessConfig {
    uint64 id;
};
@mutable @nested
struct ProcessPeriodic {

@optional StatisticVariable cpu_usage_percentage;
@optional StatisticVariable physical_memory_kb;
@optional StatisticVariable total_memory_kb;
int32 uptime_sec;
};

Table 6.11: ProcessConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Identifies the process where the service is running. The meaning of this value is platform dependent.</td>
</tr>
</tbody>
</table>

Table 6.12: ProcessPeriodic

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_usage_percentage</td>
<td>Statistic variable that provides the percentage of CPU usage of the process where the service is running. The field count of the variable contains the total CPU time that the processor spent during the publication period. Availability of this value is platform dependent.</td>
</tr>
<tr>
<td>physical_memory_kb</td>
<td>Statistic variable that provides the physical memory utilization in KiloBytes of the process where the service is running. Availability of this value is platform dependent.</td>
</tr>
<tr>
<td>total_memory_kb</td>
<td>Statistic variable that provides the virtual memory utilization in KiloBytes of the process where the service is running. Availability of this value is platform dependent.</td>
</tr>
<tr>
<td>uptime_sec</td>
<td>Time in seconds elapsed since the running service process started. Availability of this value is platform dependent.</td>
</tr>
</tbody>
</table>

Base Entity Resource Metrics
Listing 6.6: Base Entity Types

```cpp
@mutable @nested
struct EntityConfig {
    ResourceId resource_id;
    XmlString configuration;
};

@mutable @nested
struct EntityEvent{
    EntityStateKind state;
};
```

Table 6.13: EntityConfig

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource_id</td>
<td>String representation of the resource identifier associated with the entity</td>
</tr>
<tr>
<td>configuration</td>
<td>String representation of the XML configuration of the entity resource. The XML contains only children elements that are not entity resources.</td>
</tr>
</tbody>
</table>

Table 6.14: EntityEvent

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>State of the resource entity expressed as an enumeration of type EntityStateKind.</td>
</tr>
</tbody>
</table>

Network Performance Metrics
Listing 6.7: Network Performance Type

```cpp
@appendable @nested
struct NetworkPerformance {
    @optionalStatisticVariable samples_per_sec;
    @optionalStatisticVariable bytes_per_sec;
    @optionalStatisticVariable latency_millisec;
};
```

Table 6.15: NetworkPerformance

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>samples_per_sec</td>
<td>Statistic variable that provides information about the number of samples processed (received or sent) per second.</td>
</tr>
<tr>
<td>bytes_per_sec</td>
<td>Statistic variable that provides information about the number of bytes processed (received or sent) per second.</td>
</tr>
<tr>
<td>latency_millisec</td>
<td>Statistic variable that provides information about the latency in milliseconds for the data processed. The latency in a refers to the total time elapsed during the associated processing of the data, which depends on the type of application.</td>
</tr>
</tbody>
</table>
Chapter 7

Release Notes

7.1 Supported Platforms

RTI® Recording Service is supported on the platforms listed in Table 7.1.

It can also be deployed as a C++ library linked into your application. This is true for all platforms in Table 7.1 except SUSE® Linux® Enterprise Server 11.

Table 7.1: Supported Platforms in 6.0.0

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>All Linux platforms in the <em>RTI Connext DDS Core Libraries Release Notes</em> with the same version number, except Ubuntu® LTS 16.04 on Arm® v8, NI Linux 3 on Arm v7, and Wind River® Linux 7. Not supported on any custom Linux target platforms.</td>
</tr>
<tr>
<td>OS X®</td>
<td>All OS X platforms in the <em>RTI Connext DDS Core Libraries Release Notes</em> with the same version number.</td>
</tr>
<tr>
<td>QNX®</td>
<td>Only supported on the custom target platform for QNX 6.6 on x86.</td>
</tr>
<tr>
<td>Windows®</td>
<td>All Windows® platforms in the <em>RTI Connext DDS Core Libraries Release Notes</em> with the same version number.</td>
</tr>
</tbody>
</table>

7.2 Compatibility

For backward compatibility information between Recording Service 6.0.0 and previous releases, please see the Migration Guide on the RTI Community portal.

7.3 What’s New in 6.0.0

Recording Service, Replay Service, and Converter are completely new in this release. They have been entirely rewritten, with these goals in mind:

- Improving performance
- Simplifying configuration
• Becoming more consistent with other infrastructure services
• Supporting new use cases

There are a few limitations compared to the prior version:

• Does not support Recording Console, however, *RTI Admin Console* provides some of the functionality.
• Does not yet support replaying in the original partition.
• Does not include a remote shell application, but that is provided as an example.

### 7.3.1 Avoiding gaps in Replay when using time ranges

*Replay Service* used to extend the start timestamp to include relevant discovery data in the time range. While this allowed for the discovery of topics whose discovery events may have been out of the specified time range, it could also cause gaps when the discovery data and target data to be replayed were separated in time. To avoid these gaps, *Replay Service* no longer internally modifies the start timestamp defined by the user. All discovery data prior to the defined start timestamp will be discovered instantaneously.

### 7.4 What’s Fixed in 6.0.0

#### 7.4.1 Pluggable storage for Converter discarded plugin properties specified in XML

When using the pluggable storage APIs with *Converter*, the properties passed to the plugin in the *input_storage* or *output_storage* tags (as shown below) were discarded. This problem has been resolved. Now the properties are passed to the plugin.

```xml
<input_storage>
  <plugin plugin_name="StorageLib::StoragePlugin">
    <property>
      <value>
        <element>
          <name>MyProperty</name>
          <value>MyPropertyValue</value>
        </element>
      </value>
    </property>
  </plugin>
</input_storage>
```

[RTI Issue ID RECNEXTGEN-412]

### 7.5 Known Issues

#### 7.5.1 Recording Service may fail when current working directory in c:\Program Files

*Recording Service* will try to write to its database in the current working directory. If it does not have permissions to write there, it will fail with a confusing error:
This happens most commonly when running the application from within c:\Program Files on Windows systems. You can work around this by running Recording Service from a command prompt in a directory where you have write permissions.
Chapter 8

Copyrights


Trademarks

Real-Time Innovations, RTI, NDDS, RTI Data Distribution Service, DataBus, Connext, Micro DDS, the RTI logo, 1RTI and the phrase, “Your Systems. Working as one,” are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

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

© 2019 RTI