RTI Recorder

for RTI Data Distribution Service

User's Manual

Version 1.5

Printed in U.S.A. First printing.
June 2010.

Trademarks

Copy and Use Restrictions

Third-Party Copyright Notices

Technical Support

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

Contents

Chapter 1 Welcome to RTI Recorder

RTI® Recorder is an *RTI Data Distribution Service* application that records both *RTI Data Distribution Service* discovery and topic data.

The recorded data is stored in one or more files that can be viewed later. One way to view the data is with the provided SQL command-line tool, **sqlite3**. For information on displaying the recorded data, see Chapter 4: Viewing the Recorded Data.

Features

- ❏ Records data from applications in multiple domains.
- ❏ Records entire Topics, or specific Topic fields, based on POSIX file-name matching expressions.
- ❏ Records all data types except bit-fields.
- ❏ Records to multiple files with configurable file-size limits. Optionally overwrites the oldest file when the maximum number of files has been reached.
- ❏ Records the DDS SampleInfo structure and a timestamp for both discovery data and user data.
- ❏ Records using either Best Effort or Reliable communications.
- ❏ Optionally records data from only specified partitions.
- ❏ Supports remote operation.

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

Chapter 2 Configuring RTI Recorder

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

2.1 How to Load the XML Configuration

RTI Recorder loads its XML configuration from multiple locations. This section presents the various approaches, listed in load order.

The first three locations only contain QoS Profiles and are inherited from *RTI Data Distribution Service* (see Chapter 15 in the *RTI Data Distribution Service User's Manual*).

- **$NDDSHOME/resource/qos_profiles_4.5x[1]/xml/NDDS_QOS_PROFILES.xml**

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

- **File in NDDS_QOS_PROFILES**

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

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

  This file is loaded automatically if it exists.

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

The next locations are specific to *RTI Recorder*.

- **<*RTI Recorder executable location*>/../../../resource/xml/RTI_RECORDER.xml**

  This file contains the default *RTI Recorder* configuration; it is loaded if it exists. **RTI_RECORDER.xml** defines a configuration that records all topics on domain 0.

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

  This file is loaded automatically if it exists.

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

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

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

You may use a combination of the above approaches.

2.2 General Format

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

- General Properties (Section 2.3)
- Database (Output File) Properties (Section 2.4)—contained in the top-level tag, <recorder_database>
- Domain Properties (Section 2.5)—contained in the top-level tag, <domain name="String">
- TopicGroup Properties (Section 2.6)—contained in the top-level tag, <topic_group>
- RecordGroup Properties (Section 2.7)—contained in the top-level tag, <record_group>
- Remote Access Properties (Section 2.8)—contained in the top-level tag, <remote_access>
- Domain Type Configuration (Section 2.9)—contained in the top-level tag, <domain_type_config>

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

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

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

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

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

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

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

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

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

```
</dds>
```

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

- **simple_config.xml**

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

- **advanced_config.xml**

  With this configuration, *RTI Recorder* will record:

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

- **remote_shell.xml**

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

2.2.1 Configuration File Syntax

RTI Recorder follows the same XML syntax rules as *RTI Data Distribution Service.* Please see the *RTI Data Distribution Service User's Manual* for details.

2.2.2 Supported Data Types

As you will see in the following sections, each property that can appear in the configuration file uses a specific data type. *RTI Recorder* converts between the value string in the XML file and the specified type. Table 2.1 lists the supported types and the mappings used by *RTI Recorder*.

Table 2.1 **Property Value Data Types**

Type	Format	Notes
char and *octet* sequences and arrays	compact form	
DDS_Boolean	yes,1,true,on: TRUE no,0,false,off: FALSE	These values are not case sensitive.

Table 2.1 **Property Value Data Types**

Type	Format	Notes
DDS_Enum	A string. Legal values are those listed for the property in the online (HTML) documentation for the *RTI Data Distribution Service* C API.	Enum values are not case sensitive. The legal constants are those defined for the DDS C API.
DDS_Long	-2147483648 - 2147483647 0x80000000 - 0x7fffffff	A 32-bit signed integer. You may include the following unit designations: KB — 2^10 kB — 10^3 MB — 10^6 GB — 10^9 KiB — 2^10 MiB — 2^20 GiB — 2^30 For example, 100 kB is a legal value, meaning 100,000.
DDS_UnsignedLong	0 - 4294967296 0 - 0xffffffff	A 32-bit unsigned integer. You may include the following unit designations: KB — 2^10 kB — 10^3 MB — 10^6 GB — 10^9 KiB — 2^10 MiB — 2^20 GiB — 2^30 For example, 100 kB is a legal value, meaning 100,000.

Table 2.1 **Property Value Data Types**

Type	Format	Notes

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

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

2.3 General Properties

Table 2.2 describes optional properties that control *RTI Recorder*'s main module.

Table 2.2 **General Properties**

Property	Syntax	Description

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

verbosity	`<verbosity>`
	`DDS_Long`
	`</verbosity>`
		The verbosity is a bit-map that specifies what type of logging information should be printed.
		The verbosity may be:
		0: silent (both *RTI Data Distribution Service* and *RTI Recorder*)
		1: errors (both *RTI Data Distribution Service* and *RTI Recorder*) (default)
		2: warnings (*RTI Recorder* only)
		3: warnings (both *RTI Data Distribution Service* and *RTI Recorder*)
		4: information (*RTI Recorder* only)
		5: tracing (*RTI Recorder* only)
		6: tracing (both *RTI Data Distribution Service* and *RTI Recorder*)

2.4 Database (Output File) Properties

Table 2.3 describes the Database properties. All database properties are optional except database_name. All database properties must be specified within **<recorder_database>** and **</recorder_database>** tags.

Table 2.3 **Database Properties**

Property	Syntax	Description
database_name	*<database_name>* *String* *</database_name>*	**Required.** The name of the fileset used to store recorded data. *RTI Recorder* appends a set number and a segment number to the filename. Default: undefined Example: `<database_name> myfile </database_name>`
flush_ period	*<flush_period>* *DDS_Long* *</flush_period>*	Specifies how often (in seconds) to flush data to disk. Note: increasing this value causes *RTI Recorder* to use additional memory. Default: 1 second Minimum: 1 second
max_file_ segments	*<max_file_segments>* *DDS_Long* *</max_file_segments>*	Specifies how many file segments may be created. Each time the **max_file_size** limit is reached for a file segment, a new file is created if this number of segments has not been exceeded. Default: 1 Example: `<max_file_segments> 100 </max_file_segments>`
max_file_ size	<max_file_size> *FileSize* *</max_file_size>*	Specifies the maximum size for a file segment. *RTI Recorder* records data to one or more files. This property specifies the maximum file-size. Note that this is not an absolute value, but a threshold value. As soon as the threshold is exceeded, no more data is written to file. Default: 2 GB Maximum: as imposed by the operating system Example: `<max_file_size> 1 GB </max_file_size>`

Table 2.3 **Database Properties**

Property	Syntax	Description
overwrite	<overwrite> *DDS_Boolean* </overwrite>	Specifies whether or not *RTI Recorder* should delete all existing file segments in the fileset before it starts recording. This is useful if you want to reuse a data-file name between recording sessions, but do not want to keep any old data. True: if the file segments already exist, they are deleted; otherwise, the file segments are created as needed. False: if the file segments already exist, *RTI Recorder* exits; otherwise, the file segments are created as needed. Example: <name> test </name> <max_file_segments> 4 </max_file_segments> <overwrite> yes </overwrite> In this case, *RTI Recorder* will delete test_0_0, test_0_1, and test_0_2 before starting to record to test_0_0. Default:False
path_ separator	<path_separator> *DDS_Char* </path_separator>	Specifies the path separator character that *RTI Recorder* will use when creating table and column names. For instance, table names follows the "TopicName$RecordGroupName$DomainName" convention and fields in Topics uses $ to navigate hierarchical types, such as abc. '$' is used as the default path separator instead of the more conventional. ',' because $ does not require quotes when used in SQLite SQL statements. For example, to use '_' as the path separator: <path_separator> _ </path_separator> Note: this property cannot be empty.
rollover	<rollover> *DDS_Boolean* </rollover>	Specifies whether or not *RTI Recorder* should overwrite existing file segments in the fileset once the file size limit (**max_file_size**) has been reached for the last file segment. True: *RTI Recorder* overwrites existing file segments as needed (starting with the first one). False: *RTI Recorder* stops recording data. Default: False

Table 2.3 **Database Properties**

Property	Syntax	Description
segment_ number	<segment_number> *DDS_Long* </segment_number>	Specifies the first segment to use in the fileset. If the segment number is >= 0, that is the first segment number in the fileset. Default: -1. The next available segment number will be used, starting at 0. Note: the set number is determined first, then the segment number.
set_ number	<set_number> *DDS_Long* </set_number>	Specifies the set number to use in the fileset. If the set_number is >= 0, that specific fileset number is used. In this case, the <overwrite> property takes effect. Default: -1. The next available set number will be used, starting at 0.

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

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

RTI Recorder uses a fixed file-naming scheme:

name_set-number_segment-number

Where:

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

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

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

2.5 Domain Properties

Table 2.4 describes the Domain properties. All Domain properties are optional.

Table 2.4 **Domain Properties**

Property	Syntax	Description

deserialize_mode	`<deserialize_mode>`
	`DDS_Enum`
	`</deserialize_mode>`
		Determines how topic data is stored in a database (serialized or deserialized).
		The following values are allowed:
		❏ RTIDDS_DESERIALIZEMODE_AUTOMATIC —deserialize data if possible, otherwise store data in serialized format.
		❏ RTIDDS_DESERIALIZEMODE_NEVER —Do not deserialize the data; store data in serialized format.
		❏ RTIDDS_DESERIALIZEMODE_ALWAYS —Only store data if it can be deserialized first.
		Default: RTIDDS_DESERIALIZEMODE_NEVER
		See Recording Large User Data Types (Section 2.5.1).

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

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

Domain properties must be specified inside **<domain name="String">** and **</domain>** tags. If you want to use a RecordGroup (Section 2.7), you must assign a domain name with these tags, even if you do not specify any domain properties (because the domain name is needed in the RecordGroup's domain_ref property).

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

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

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

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

2.5.1 Recording Large User Data Types

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

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

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

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

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

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

2.6 TopicGroup Properties

A TopicGroup is an *optional* logical collection of Topics. If you are not going to have a RecordGroup in the configuration file, you do not need a TopicGroup. (See Section 2.7.)

Table 2.5 describes the TopicGroup properties. The following properties are required:

- ❏ field_expr
- ❏ shared_table
- ❏ topics

Table 2.5 **TopicGroup Properties**

Property	Syntax	Description
auto_ detect_ reliability	`<auto_detect_reliability>` *DDS_Boolean* `</auto_detect_reliability>`	If set to true, use the same reliability as the Publisher of the matched Topic. Default: false.
compact_ char_array	`<compact_char_array>` *DDS_Boolean* `</compact_char_array>`	Store array of char in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large. Default: true.
compact_ octet_array	`<compact_octet_array>` *DDS_Boolean* `</compact_octet_array>`	Store array of octet in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large. Default: true.
compact_ char_ sequence	`<compact_char_sequence>` *DDS_Boolean* `</compact_char_sequence>`	Store sequence of char in a single column. The default (true) saves the most space. While it is possible to store individual elements in separate columns, it is not recommended as the number of columns stored can become very large. Default: true.

Table 2.5 **TopicGroup Properties**

Property	Syntax	Description

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

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

field_expr	`<field_expr>`
	`  *POSIX fn expressions*`
	`</field_expr>`
		Required.
		A list of comma-separated POSIX expressions that specify which fields in the Topics to record. (The Topics are specified with <topics>, see Table 2.6.)
		This parameter is ignored when recording serialized data.

Table 2.5 **TopicGroup Properties**

Property	Syntax	Description
shared_table	<shared_table> *DDS_Boolean* </shared_table>	**Required.** Specifies whether the tables of recorded data are shared or exclusive. *RTI Recorder* stores Topic data in tables; those tables can be Shared or Exclusive. An Exclusive table means that each Topic recorded in a RecordGroup is stored in its own table. The name of the table follows the convention: `TopicName$RecordGroupName$DomainName` (The '$' separator can be changed with the path_separator database property.) Thus, two topics with the same name but from two different TopicGroups are stored in separate tables. A shared table means that Topics with the same name are stored in the same table, regardless of where it was recorded from. In this case the table has an additional column, 'table_prefix', which stores the table prefix in the form: RecordGroupName$DomainName. Default: False (exclusive).
topics	<topics> *POSIX fn expressions* </topics>	**Required.** Specifies a topic expression and any exemptions to that expression. See Table 2.6.

Table 2.6 **Topics Properties**

Property	Syntax	Description
exemption	<exemption> *POSIX fn expressions* </exemption>	Specifies a comma-separated list of expressions that should *not* be recorded. Default: nothing is exempt.
topic_expr	<topic_expr> *POSIX fn expression* </topic_expr>	**Required.** A POSIX expression that specifies the names of the Topics to be included in the TopicGroup. The syntax and semantics are the same as for Partition matching. Default: null.

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

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

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

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

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

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

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

Note: Topics are never removed from a TopicGroup. The resources used to create DataReaders for discovered Topics are not released if/when the Topics are deleted.

Note: *RTI Recorder* will ignore Topics published with a type that conflicts with an already discovered type.

2.7 RecordGroup Properties

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

RecordGroups are optional. If you do not create one, *RTI Recorder* will not record any data. This is useful if you want to start *RTI Recorder* in "stand-by" mode—then you can use remote access (see Section 2.8) to switch to a different configuration file (one that does have a RecordGroup) and start recording.

Table 2.7 describes the RecordGroup properties. The following properties are required:

- ❏ domain_ref
- ❏ topic_ref

Table 2.7 **RecordGroup Properties**

Property	Syntax	Description

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

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

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

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

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

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

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

2.8 Remote Access Properties

As you will see in Chapter 7: Accessing RTI Recorder from a Remote Location, you can create an *RTI Data Distribution Service* application that can remotely control *RTI Recorder*.

By default, Remote Access is turned off in *RTI Recorder* for security reasons.

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

The remote application can send commands to *RTI Recorder* that will:

- ❏ Start/stop recording.
- ❏ Shutdown *RTI Recorder*.
- ❏ Reconfigure *RTI Recorder*.

Table 2.8 describes the Remote Access properties. All Remote Access properties must be specified inside **<remote_access>** and **</remote_access>** tags. All remote access properties are optional unless otherwise noted.

Table 2.8 **Remote Access Properties**

Property	Syntax	Description
accept_ broadcast_ commands	<accept_broadcast_commands> *DDS_Boolean* </accept_broadcast_commands>	Specifies if *RTI Recorder* will accept commands that have been broadcast to any *RTI Recorder* or only accept commands addressed specifically to it. (See Chapter 7: Accessing RTI Recorder from a Remote Location.) Default: true
enabled	<enabled> *DDS_Boolean* </enabled>	Enables or disables remote access to *RTI Recorder* from another application. (See Chapter 7: Accessing RTI Recorder from a Remote Location.) Default: false (remote access disabled)
datareader_qos	<datareader_qos> *DDS_DataReaderQos* </datareader_qos>	Configures the QoS for the DataReader created by *RTI Recorder's* Remote Access module. Default: default DataReader QoS settings.
datawriter_qos	<datawriter_qos> *DDS_DataWriterQos* </datawriter_qos>	Configures the QoS for the DataWriter created by *RTI Recorder's* Remote Access module. Default: default DataWriter QoS settings.
publish_status_ period	<publish_status_period> *DDS_Long* </publish_status_period>	Specifies, in seconds, the period between each status message sent by *RTI Recorder*. Default: 1. Minimum value: 1.
publisher_qos	<publisher_qos> *DDS_PublisherQos* </publisher_qos>	Configures the QoS for the Publisher created by *RTI Recorder's* Remote Access module. Default: default Publisher QoS settings.
remote_access_ domain	<remote_access_domain> *String* </remote_access_domain>	**Required if** enabled **is true.** Specifies which domain *RTI Recorder* will use to enable remote access. Only one domain can be specified. Note that this is a String, not a Domain ID. It is the same String used in the <domain name="String"> </domain> line. Default: false
subscriber_qos	<subscriber_qos> *DDS_QosPolicy* </subscriber_qos>	Configures the QoS for the Subscriber created by *RTI Recorder's* Remote Access module. Default: default Subscriber QoS settings.

2.9 Domain Type Configuration

The Domain Type Config allows the users to pass DDS type configuration information to *RTI Recorder* and *RTI Converter* in the form of XML type-configuration files.

Table 2.9 describes the Domain Type Config properties. All Domain Type Config properties are optional.

Table 2.9 **Recorder Type Configuration Properties**

Property	Syntax	Description
domain_type_config	<domain_type_config> *Domain Type Config Properties* </domain_type_config>	Type configuration for specific domains. See Table 2.10, "Domain Type Configuration Properties".

Table 2.10 **Domain Type Configuration Properties**

Property	Syntax	Description
domain_group	<domain_group> <element> *Domain Group Properties* </element> </domain_group>	A list of type configuration elements. The elements tag can be repeated. See Table 2.11, "Domain Group Type Configuration Properties".

Table 2.11 **Domain Group Type Configuration Properties**

Property	Syntax	Description
domain_filter	<domain_filter> <element> *POSIX fn expression* </element> </domain_filter>	A list of expressions for the domains for which these Domain Group Properties should apply. The element tag can be repeated. See Table 2.12, "Domain Filter Type Configuration Properties".
type_config	<type_config> *XML Properties* </type_config>	The type configuration for this Domain Group. See Table 2.13, "Type Properties"s.

Table 2.12 **Domain Filter Type Configuration Properties**

Property	Syntax	Description
xml	<xml> *XML Type Config. Properties* </xml>	The XML type configuration for this domain group. See Table 2.14, "XML Type Properties".

Table 2.13 **Type Properties**

Property	Syntax	Description
register_top_level	<register_top_level> *Boolean* </register_top_level>	Whether or not to register the top-level types with their canonical names
max_string	<max_string> *Integer* </max_string>	The default values to use when there are unbounded strings in a type.
max_sequence	<max_sequence> *Integer* </max_sequence>	The default values to use when there are unbounded sequences in a type.
path	<path> <element> *Path* </element> </path>	A list of the paths to be used when searching for XML type-configuration paths. The *element* tag can be repeated.
file_group	<file_group> <element> *File Group Properties* </element> <file_group>	A list of file groups associated with this domain group. A file group is parsed into a single Document Object Module. The element tag can be repeated. See Table 2.14, "XML Type Properties".

Table 2.14 **XML Type Properties**

Property	Syntax	Description
register_top_level	<register_top_level> *Boolean* </register_top_level>	Whether or not to register the top-level types with their canonical names. This overrides the parent
max_string	<max_string> *Integer* </max_string>	The default values to use when there are unbounded strings in a type. This overrides the parent.
max_sequence	<max_sequence> *Integer* </max_sequence>	The default values to use when there are unbounded sequences in a type.This overrides the parent.
file_name	<file_name> <element> *File Name* </element> </file_name>	A list of the files that contain XML type-definitions. The element tag can be repeated.

Chapter 3 Using RTI Recorder

This chapter describes how to start and stop *RTI Recorder*.

3.1 Starting RTI Recorder

RTI Recorder's executable is in **<*installation directory*>/target/bin/<*architecture*>**.

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

To run the tool you must select a configuration name with **-cfgName**.

For example:

```
$ cd rtirecorder.1.5.x-ndds.4.5y/target/bin/i86Linux2.4gcc3.2.2
$ rtirecorder -cfgName default
```

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

```
$ cd rtirecorder.1.5.x-ndds.4.5y/target/bin/i86Linux2.4gcc3.2.2
$ rtirecorder -listCfgs
```

To use your own configuration file:

```
$ cd rtirecorder.1.5.x-ndds.4.5y/target/bin/i86Linux2.4gcc3.2.2
$ rtirecorder -cfgFile config-file.xml -cfgName my_recorder_cfg
```

Table 3.1 describes the command-line options. Chapter 2: Configuring RTI Recorder describes the contents of the configuration file. Example files are provided in the **<installation directory>/examples** directory.

Table 3.1 **RTI Recorder's Command-Line Options**

Command-line Option	Description
`-appName <name>`	Assigns an application name to the DomainParticipants created by *RTI Recorder*. If not specified, the same name used in **-cfgName** will be used.
`-cfgFile <file>`	Specifies the XML configuration file (path and filename). In addition to the file provided using this command-line option, *RTI Recorder* can load other XML files—see Section 2.1.
`-cfgName <name>`	**Required.** This name is used to find the matching **<recorder>** tag in the configuration file.
`-dbName`	Names the fileset to use for storing recorded data. Default: **rtirecorder.dat**
`-help`	Prints version information and list of command-line options.
`-licenseFile <file>`	Specifies the license file (path and filename). Only applicable to licensed versions of *RTI Recorder*. If not specified, *RTI Recorder* looks for the license as described in Section 2.3 in the *Getting Started Guide*.
`-listCfgs`	Lists the available configuration profiles.
`-verbosity`	Specifies what type of logging information should be printed. 0: silent (both RTI Data Distribution Service and RTI Recorder) 1: errors (both RTI Data Distribution Service and RTI Recorder) (default) 2: warnings (RTI Recorder only) 3: warnings (both RTI Data Distribution Service and RTI Recorder) 4: information (RTI Recorder only) 5: tracing (RTI Recorder only) 6: tracing (both RTI Data Distribution Service and RTI Recorder) This property can also be set in the configuration file. However, this command-line option overrides the value specified in the configuration file.

3.2 Stopping RTI Recorder

To stop RTI Recorder: press **Ctrl-C**. *RTI Recorder* will close all files and perform a clean shutdown.

You can also start, stop, and even reconfigure *RTI Recorder* remotely—see Chapter 7: Accessing RTI Recorder from a Remote Location.

Chapter 4 Viewing the Recorded Data

RTI Recorder stores data in a SQL database. This chapter describes how the data is stored and how to view the data. You can view all the data recorded by *RTI Recorder* with **sqlite3**, a SQL command-line tool.

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

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

```
$ cd rtirecorder.1.5.x-ndds.4.5y/host/bin/i86Linux2.4gcc3.2.2
$ sqlite3 <recorded file>
```

Then you can list all the available topics by entering:

```
sqlite> .tables
```

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

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

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

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

Note: For more example commands, please see the Chapter 3 in the Getting Started Guide.

To exit **sqlite3**, enter:

```
sqlite> .exit
```

4.1 Format of the Recorded Data

4.1.1 Discovery Data

RTI Recorder stores discovery-related data in these tables:

- ❏ DCPSParticipant — corresponds to the Participant Built-in Topic
- ❏ DCPSPublication — corresponds to the Publication Built-in Topic
- ❏ DCPSSubscription — corresponds to the Subscription Built-in Topic

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

RTI Recorder stores the following information, in this order:

- ❏ A timestamp (in microseconds since Jan. 1st, 1970).
- ❏ State (internal information, can be ignored)
- ❏ Type (internal information, can be ignored)
- ❏ The domain ID from which the sample was received.
- ❏ The BuiltinTopicData.
- ❏ The SampleInfo information, as described in the *RTI Data Distribution Service* documentation.

4.1.2 User Data

- ❏ Deserialized Data

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

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

```
struct Bar {
    long x;
};

struct Foo {
    Bar bar;
};
```

❑ Topic Data

RTI Recorder creates a table called "TopicName$RecordGroupName$DomainName" for each recorded topic (unless the shared_table property is true).

RTI Recorder stores topic data as specified in the subscription properties.

For each topic, *RTI Recorder* also stores the following, in this order:

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

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

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

4.1.3 Other Tables

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

Chapter 5 Exporting Recorded Data

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

The executable, **rtirecconv**, is located in **<*install-dir*>/host/bin/<*architecture*>**.

To see a list of available options, enter:

```
$ rtirecconv -help
```

You will see the following:

```
rtirecconv [options] fileset
Options:
-verbosity <mask>  - verbosity mask
-compact <mode>    - How to read octet/char arrays and sequences. Default:auto
                     auto - Detect automatically how data is stored and
                            compact/do not compact each type of data accordingly.
                     yes - Show the structures with compact data regardless of
                            how they are stored.
                     no - Do not compact the structures regardless of how they
                            are stored.
-tableExpr         - POSIX fn-name expressions, Can be repeated.
-includeInfo       - Include the DDS_SampleInfo structure for each sample
-includeDiscovery  - Include discovery traffic
-includeNonData    - Include non-data samples
-decodeUnknown     - Try to decode samples from DDS_DataWriters with no typecode
-format <format>   - Output format. Default: xml
                      xml  - Output in XML format
                      csv  - Output CSV format
                      html - Output in HTML table format
                      sql  - Output in SQL table format
```

```
-decodeChar <format> - Decode char data in this format. Default: text
                       hex  - Decode char data as hex
                       text - Decode char data as ASCII text
-decodeOctet <format>   - Decode octet data in this format. Default: hex
                          hex  - Decode octet data as hex
                          text - Decode octet data as ASCII text
-time <format>          - Format for timestamp
                          epoch - Show timestamp in microsec from January 1 1970
                          gmt   - Show timestamp in GMT
-filePrefix <prefix>    - Create one file per table called <prefix>_<Table>
-outputFile <file>      - Name of output file, cannot be used with -filePrefix
```

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

```
$ rtirecconv mydata.dat_0
```

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

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

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

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

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

Chapter 6 Example Configuration Files

This chapter shows how to configure *RTI Recorder* for a variety of situations:

6.1 How to Record All Topics in a Single Domain

Scenario

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

Configuration File

```
<dds>
<recorder name="scenario1">
    <recorder_database>
        <database_name> mydomaindata </database_name>
        <max_file_size> 1 GB </max_file_size>
    </recorder_database>
    <domain name="mydomain">
        <domain_id> 54 </domain_id>
    </domain>
```

```
        <topic_group name="All">
            <topics>
               <topic_expr> * </topic_expr>
            </topics>
            <field_expr> * </field_expr>
        </topic_group>
        <record_group name="sub0">
            <domain_ref>
               <element> mydomain </element>
            </domain_ref>
            <topic_ref>
               </element> All </element>
            </topic_ref>
        </record_group>
    </recorder>
    </dds>
```

Expected Outcome

Since by default, *RTI Recorder* starts recording automatically, the expected outcome is a single file about 4 GB with all the data in a file called **mydomaindata_0_0**. By default, *RTI Recorder* will store deserialized data, so the file will have one column for each field in the topic.

6.2 How To Record a Subset of Data from Multiple Domains

Scenario

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

Configuration File

```
    <dds>
    <recorder name="scenario2">
        <recorder_database>
            <database_name> mydomaindata </database_name>
            <max_file_size> 1 GB </max_file_size>
        </recorder_database>
```

```
    <domain name="mydomain54">
        <domain_id> 54 </domain_id>
    </domain>
    <domain name="mydomain98">
        <domain_id> 98 </domain_id>
    </domain>
    <topic_group name="Sensor">
        <topics>
           <topic_expr> Sensor* </topic_expr>
        </topics>
        <field_expr> *value* </field_expr>
    </topic_group>
    <record_group name="sub0">
        <domain_ref>
           <element> mydomain54 </element>
        </domain_ref>
        <topic_ref>
           </element> All </element>
        </topic_ref>
    </record_group>
    <record_group name="sub1">
        <domain_ref>
           <element> mydomain98 </element>
        </domain_ref>
        <topic_ref>
           </element> All </element>
        </topic_ref>
    </record_group>
</recorder>
</dds>
```

Expected Outcome

Since by default *RTI Recorder* starts recording to file automatically, the expected outcome is a single file about 4 GB, with all the data in a file called mydomaindata_0_0. By default, *RTI Recorder* will store deserialized data; so the file will have one column per field in the topic.

6.3 How To Record Data to Multiple Files

Scenario

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

Configuration File

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

Expected Outcome

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

6.4 How To Record Serialized Data

Scenario

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

Configuration File

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

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

Expected Outcome

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

6.5 How To Record Using Best-Effort Reliability

Scenario

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

Configuration File

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

Expected Outcome

RTI Recorder will use DataReaders with best-effort Reliability to record all data.

6.6 How To Enable Remote Access

Scenario

RTI Recorder is part of a larger system that must reach a steady state before it starts recording. *RTI Recorder* should use domain ID 54 and partition "rti" for communication with the controller.

Configuration File

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

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

Expected Outcome

RTI Recorder will communicate with a remote controller on domain ID 54 using partition "rti." Status information will be published every 10 seconds.

Chapter 7 Accessing RTI Recorder from a Remote Location

Perhaps you want to start/stop *RTI Recorder* from another machine, or even reconfigure it to change what is being recorded. You can create an *RTI Data Distribution Service* application that can remotely control *RTI Recorder*. This chapter explains how.

To control *RTI Recorder* from a remote location:

1. Configure *RTI Recorder* to allow remote access (see Remote Access Properties (Section 2.8).
2. Create an *RTI Data Distribution Service* application using the provided **rtirecorder.idl** file. You will use the *RTI Data Distribution Service rtiddsgen* tool to generate the basics, and then add code to send your desired remote commands.

7.1 Overview

If *RTI Recorder* is configured to allow remote access, it creates a DataReader for a "command" Topic (named RTI_RECORDER_CMD_TOPIC) and a DataWriter for "status" Topic (named RTI_RECORDER_STATUS_TOPIC). So *RTI Recorder* will write status updates and read commands.

These topics' types and names are specified in the IDL file, **recorder.idl** (provided in the **examples** directory).

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

Your remote-access application will use the following constants:

- **RTI_REMOTECTX_MSG_TYPE** Register a type of this name, as seen in Figure 7.1.
- **RTI_RECORDER_CMD_TOPIC** Create a DataWriter with this Topic name, as seen in Figure 7.2.
- **RTI_RECORDER_STATUS_TOPIC** Create a DataReader with this Topic name, as seen in Figure 7.2.

See Remote Control Messages (Section 7.3) for more information.

Figure 7.1 **Registering the Message Type**

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

7.2 Establishing a Connection with RTI Recorder

To establish a connection with *RTI Recorder*, your remote-access application needs:

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

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

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

Figure 7.2 shows how to create the Entities in your remote-control application using the C API. (A general knowledge of DDS is assumed.)

Figure 7.2 **Creating the Required DDS Entities**

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

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

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

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

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

7.3 Remote Control Messages

RTI Recorder exchanges messages with your remote-access application by publishing and subscribing to two special remote-access topics. Both topics use the same message format, shown in Figure 7.3.

Figure 7.3 **Top Level Structure for Remote Control Messages**

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

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

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

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

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

msg — A union of different message types. The discriminator must be set to one of the message types listed in Table 7.1.

The code fragment in Figure 7.4 shows how to set the message type in the remote-access application.

Figure 7.4 **Assigning a Message Type (C Language)**

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

Depending on the message type, the correct union member must also be filled in. For example, Figure 7.5 shows how to construct a message to *RTI Recorder* to read a new configuration from a file. In this example, the new configuration is to be read from a file on the same file-system as *RTI Recorder*.

Figure 7.5 **Sending a Command to RTI Recorder to Read a New Configuration File**

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

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

/* copy the filename of the file RTI Recorder shall read from */
strncpy(msg->msg._u.config.config_from_string,filename,512);

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

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

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

Table 7.1 **Messages Types Exchanged Between RTI Recorder and a Remote Access Application**

Direction	Message Type (add prefix: RTI_REMOTECTX_ MSG_)	Description
From: your *RTI Data Distribution Service* Remote-Control Application To: *RTI Recorder*	RECORDER_START	Instructs RTI Recorder to start recording.
	RECORDER _STOP	Instructs RTI Recorder to stop recording.
	RECORDER_CONFIGURE	Instructs RTI Recorder to reconfigure according to the contents of the message. Stop RTI Recorder before sending this message: If RTI Recorder has already been stopped, it will read the new configuration and restart. It will not automatically start recording unless auto_start (see Table 2.2, "General Properties") is true (the default case). If RTI Recorder has not already been stopped, an error is returned.
	RECORDER_SHUTDOWN	Instructs RTI Recorder to shutdown and exit.
	RECORDER_ADD	Instructs RTI Recorder to add entities based on the contents of the message.
	RECORDER_UPDATE	Instructs RTI Recorder to update entities based on the contents of the message.
	RECORDER_MODIFY	Instructs RTI Recorder to add OR modify entities based on the contents of the message.
	RECORDER_DELETE	Instructs RTI Recorder to delete entities based on the contents of the message.
	RECORDER_PAUSE	Instructs RTI Recorder to pause entities based on the contents of the message.
	RECORDER_RESUME	Instructs RTI Recorder to resume recording of previously paused entities based on the contents of the message.
	RECORDER_PING	Instructs RTI Recorder to send the recording model [a] to the Remote Control application.
To: your *RTI Data Distribution Service* Remote Control Application From: *RTI Recorder*	RECORDER_INFO	When RTI Recorder publishes statistics, it periodically sends out this message type.
	RECORDER_RESPONSE	Indicates that this message is a response to a command.

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

7.4 Using the Example Remote-Access Application—RTI Recorder Shell

RTI Recorder Shell is an *RTI Data Distribution Service* application that can remotely control *RTI Recorder*. You can use it to start, stop, and reconfigure *RTI Recorder*. *RTI Recorder Shell* is not meant as a complete solution to remotely controlling *RTI Recorder*. Its purpose is just to give you an idea of what can be done.

RTI Recorder Shell, **rtirecsh**, is in ***<installation directory>*/host/bin/*<architecture>***.

For example, to start *RTI Recorder Shell*, enter:

```
cd <installation directory>/host/bin/<your architecture>
rtirecsh -domain <domain ID>
```

Table 7.2 lists the command-line options you can use when starting *RTI Recorder Shell*. Once it is running, you can use the commands described in RTI Recorder Shell's Commands (Section 7.4.1).

Table 7.2 **RTI Recorder Shell's Command-Line Options**

Command-line Option	Description
`-domain <domain ID>`	**Required.** Specifies the domain ID. The <domain ID> is an integer between 0 and 99.
`-partition <names>`	Specifies an optional, comma-separated list of partition names. This option is necessary if *RTI Recorder* is configured to enable remote access in a particular partition.
`-noudpv4`	Disables the UPDv4 transport.
`-updv6`	Enables the UDPv6 transport.
`-shmem`	Enables the shared memory transport.
`-multicast`	Enables multicast.

Table 7.2 **RTI Recorder Shell's Command-Line Options**

Command-line Option	Description
`-verbosity <mask>`	The verbosity is a bit-map that specifies what type of logging information should be printed. The verbosity may be: 0 — No messages 1 — Exceptions (default) 2 — Warnings 4 — Information 7 — All types
`-help`	Prints version information and a list of options.

7.4.1 RTI Recorder Shell's Commands

- ❏ add (Section 7.4.1.1)
- ❏ configure (Section 7.4.1.2)
- ❏ delete (Section 7.4.1.3)
- ❏ exit (Section 7.4.1.4)
- ❏ info (Section 7.4.1.5)
- ❏ model (Section 7.4.1.6)
- ❏ modify (Section 7.4.1.7)
- ❏ pause (Section 7.4.1.8)
- ❏ resume (Section 7.4.1.9)
- ❏ shutdown (Section 7.4.1.10)
- ❏ start (Section 7.4.1.11)
- ❏ status (Section 7.4.1.12)
- ❏ stop (Section 7.4.1.13)
- ❏ update (Section 7.4.1.14)

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

- ❏ The configuration model: the XML configuration (similar to the XML configuration file used to configure *RTI Recorder*.)

- The run-time model: an XML description of the DDS entities that have been created based on the configuration.

The format of this XML is the same as the configuration format for *RTI Recorder* (see Chapter 2: Configuring RTI Recorder). The top-level tag must be <dds> (followed by <recorder>) or just <recorder>.

Some examples for <model> are:

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

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


7.4.1.1 add

This command adds entities to *RTI Recorder*.

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

```
add <model>
```

7.4.1.2 configure

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

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

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

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

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

- **-localfile <filename>**

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

  ```
  RTI Recorder Shell> stop
  RTI Recorder Shell> configure myrecorder -localfile myconfig.xml
  ```

  RTI Recorder Shell will read the contents of **myconfig.xml** and send it to *RTI Recorder*, which will search for a tag <recorder name="myrecorder">. If auto_start (see Table 2.2, "General Properties") is true (the default case), it is not necessary to run the start command to start *RTI Recorder.* If auto_start is false in the new configuration, then issue the start command in *RTI Recorder Shell* to start recording:

  ```
  RTI Recorder Shell> start
  ```

- **-remotefile <filename>**

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

  Example: Assume that you want reconfigure *RTI Recorder* with a file called **remotemyconfig.xml**, which resides on the same file-system as *RTI Recorder*.

```
RTI Recorder Shell> stop
RTI Recorder Shell> configure -myrecorder -remotefile remotemyconfig.xml
```

  RTI Recorder will read the contents of **remotemyconfig.xml** and reconfigure with the contents of the tag <recorder name="myrecorder">. Depending on the configuration file, it may be necessary to start it:

```
RTI Recorder Shell> start
```

7.4.1.3 delete

This command deletes entities from *RTI Recorder.*

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

```
delete <model>
```

7.4.1.4 exit

This command exits *RTI Recorder Shell*.

```
RTI Recorder Shell> exit
```

7.4.1.5 info

This command shows which *RTI Recorder* session the *RTI Recorder Shell* is connected to. The output looks similar to this:

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

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

7.4.1.6 model

This command prints the current model of *RTI Recorder.*

```
RTI Recorder Shell> model
```

7.4.1.7 modify

This command adds new entities to *RTI Recorder,* or updates existing entities.

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

```
modify <model>
```

7.4.1.8 pause

This command pauses the recording of entities in *RTI Recorder.*

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

```
pause <model>
```

7.4.1.9 resume

This command resumes the recording of already paused entities in *RTI Recorder.*

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

```
resume <model>
```

7.4.1.10 shutdown

This command causes *RTI Recorder* to shut down and terminate.

This command can only be issued when *RTI Recorder* has been stopped.

7.4.1.11 start

The **start** command is used to start *RTI Recorder*. Note that this command only works after stopping *RTI Recorder* first, since *RTI Recorder* is by definition started when it is launched.

When the start command is given, *RTI Recorder* will shut down completely, delete all state and objects and start from scratch. By default, *RTI Recorder* will create a new fileset each time it is started.

7.4.1.12 status

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

```
RTI Recorder Shell> status
```

The output is similar to the following:

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

- **Version** *RTI Recorder*'s version
- **Timestamp** The timestamp on *RTI Recorder* when status message was sent.
- **State** *RTI Recorder's* state. The following states are possible:
  - IDLE
  - RECORDING
  - STOPPED (*RTI Recorder* has been stopped and is not recording any user data)
  - RECONFIGURE
  - SHUTDOWN
  - RESTART
  - DOWNLOAD (*RTI Recorder* is downloading a new configuration)
- **Config file** The name of the file from which *RTI Recorder* read its configuration. If the configuration was received with **configure -localfile**, this field is not available.

- **Database file** The file-segment currently being written to.
- **Received bytes** Total amount of data that has been written to file.
- **Saved bytes** Total number of data that has is currently saved to file. Note that if the rollover property is true, then Saved bytes may be less than Received bytes.

7.4.1.13 stop

This command stops *RTI Recorder* from recording user data.

```
RTI Recorder Shell> stop
```

7.4.1.14 update

This command updates entities in *RTI Recorder*.

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

```
update <model>
```

7.4.2 Running More than One RTI Recorder in the Same Domain

RTI Recorder Shell can only keep track of one *RTI Recorder*. To control multiple R*TI Recorder* sessions in the same domain with *RTI Recorder Shell*, run each *RTI Recorder* session in a separate partition.

For the first instance of *RTI Recorder*, change the configuration file as follows:

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

For the second instance of *RTI Recorder*, change the configuration file as follows:

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

Then you can run the *RTI Recorder Shell* for each partition:

```
rtirecsh -partition RecorderA
rtirecsh -partition RecorderB
```