RTI Persistence Service Release Notes

Version 7.3.0

Contents

1 Copyrights and Notices **1**

2 Supported Platforms **3**

3 Compatibility **4**

4 What's New in 7.3.0 LTS **5**

4.1 Performance improvement for durable writer history and Persistence Service with high late-joiner activity . . . 5

4.2 Ability to filter by related source GUID using new <propagate_related_source_guid> tag . . . 5

5 What's Fixed in 7.3.0 LTS **6**

5.1 Crashes . . . 6

5.1.1 [Critical] Persistence Service could potentially crash if a Durable Subscription name was longer than 512 characters . . . 6

5.1.2 [Critical] Crash in Persistence Service when using ContentFilteredTopics and sharing type codes for discovery . . . 6

5.2 Memory Leaks/Growth . . . 6

5.2.1 [Major] Persistence Service could leak a file handle on Windows when purging sample logs upon startup . . . 6

5.3 Other . . . 7

5.3.1 [Critical] Configuring persistence group's DataWriter QoS to use compression and XCDR2 encapsulation caused samples to not be received from Persistence Service . 7

5.3.2 [Critical] <content_filter> tag caused segmentation fault in Persistence Service if filter expression was invalid . . . 8

5.3.3 [Critical] DataReader stopped receiving samples from Persistence Service DataWriter that uses DDS fragmentation when Persistence Service ran in PERSISTENT mode . . . 8

5.3.4 [Critical] Error when receiving compressed batch samples . . . 8

5.3.5 [Minor] <configuration_variables> tag not supported in Persistence Service . . . 9

5.3.6 [Minor] Memory Leak when parsing environment variable in XML that does not exist 9

6 Previous Releases **10**

6.1 What's New in 7.2.0 . . . 10

6.1.1 Persistence Service compatible with Monitoring Library 2.0 . . . 10

6.1.2 Support for dynamic certificate revocation and renewal without needing to restart Persistence Service . . . 11

6.1.3 New Library API functions in C allow get/set QoS operations on internal DomainParticipants . . . 11
6.1.4 New Library API functions in C allow get/set QoS operations on internal Publisher/Subscriber entities . . . 11
6.1.5 Persistence Service now allows you to set rtps_app_id, giving you greater control over value of DomainParticipant's GUID prefix . . . 12
6.2 What's Fixed in 7.2.0 . . . 12
6.2.1 [Critical] Segmentation fault issue with Persistence Service and Distributed Logger . . . 12
6.2.2 [Major] Error creating a persistence group DataWriter when setting dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size to -1 . . . 13
6.2.3 [Minor] <reader_checkpoint_frequency> may not have been applied correctly . . . 13
6.3 What's New in 7.1.0 . . . 13
6.3.1 Persistence Service support as a library for all supported architectures . . . 13
6.3.2 Removed ability to share a database connection in Persistence Service and durable writer history . . . 14
6.3.3 Third-party software upgrade . . . 14
6.4 What's Fixed in 7.1.0 . . . 14
6.4.1 [Major] Persistence Service stored/forwarded samples multiple times when there were two or more equivalent versions of a type for a Topic . . . 14
6.4.2 [Major] Unexpected fatal error when number of instances reached the limit * . . . 15
6.4.3 [Minor] Persistence Service XSD schema was broken * . . . 15
6.4.4 Fixes related to vulnerabilities . . . 15
[Critical] Potential arbitrary SQL query execution when enabling database locking . . . 15
6.5 What's New in 7.0.0 . . . 16
6.5.1 Support for external databases is discontinued . . . 16
6.5.2 Default journal_mode and synchronization changed to WAL and NORMAL, respectively . . . 16
6.5.3 Third-party software upgrade . . . 17
6.6 What's Fixed in 7.0.0 . . . 17
6.6.1 [Critical] Fatal error when persisting unkeyed Topics upon restore or IP mobility event 17
6.6.2 [Major] Samples published out of order from same virtual GUID were dropped . . . 18
6.6.3 [Minor] Schema files not compliant with DDS-XML specification . . . 18

7 Known Issues 19
7.1 Coherent Changes not Propagated as Coherent Set . . . 19
7.2 TopicQueries not Supported in PERSISTENT Mode . . . 19
7.3 <comm_ports> not Supported when Using Real-Time WAN Transport . . . 19
7.4 Persistence Service DataReaders Ignore Serialized Key Propagated with Dispose Updates . . . 20
7.5 Synchronizing Data Samples between Persistence Service Instances Can Consume Significant Bandwidth . . . 20
7.6 Some tags in the XML configuration must be grouped in a strict order . . . 20

8 Available Documentation 21

Chapter 1 Copyrights and Notices

Trademarks

Copy and Use Restrictions

Third-Party Software

Notices

Deprecations and Removals

Any deprecations or removals noted in this document serve as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software.

Deprecated means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in a release,

RTI hereby provides customer notice that RTI reserves the right after one year from the date of such release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

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/

Chapter 2 Supported Platforms

RTI® Persistence Service is included with *RTI Connext*. If you choose to use it, it must be installed on top of *Connext* with the same version number.

See Supported Platforms, in the RTI Connext Core Libraries Release Notes.

RTI tests *Persistence Service* with a file-system only, using PERSISTENT mode.

Chapter 3 Compatibility

Note: For backward-compatibility information between this and previous releases, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

Chapter 4 What's New in 7.3.0 LTS

This section describes new features in *Connext* 7.3.0 LTS, compared to 7.2.0. For information on new features in releases 7.0.0, 7.1.0, and 7.2.0, which are all also part of 7.3.0 LTS, see *Previous Releases*.

4.1 Performance improvement for durable writer history and Persistence Service with high late-joiner activity

You may have experienced performance degradation while using durable writer history and *Persistence Service*, particularly in scenarios with high late-joiner activity. This problem was due to the durable *DataWriter* not caching repair samples in the durable writer history, leading to excessive database queries from the different late-joiners. *Connext* now caches repair samples if the cache has sufficient capacity (`writer_qos.resource_limits.max_samples` is less than or equal to `writer_qos.durability.storage_settings.writer_instance_cache_allocation.max_count`).

4.2 Ability to filter by related source GUID using new <propagate_related_source_guid> tag

This release introduces a new tag called <propagate_related_source_guid> under <persistence_group> that allows propagating and storing the related source GUID associated with the incoming samples to *Persistence Service*. Propagating the related source GUID makes it possible for a *DataReader* receiving samples from *Persistence Service* to create a ContentFilteredTopic on this field.

See Creating Persistence Groups for information on the new tag. See SQL Filter Expression Notation for information on filtering metadata.

Chapter 5 What's Fixed in 7.3.0 LTS

5.1 Crashes

5.1.1 [Critical] Persistence Service could potentially crash if a Durable Subscription name was longer than 512 characters

Persistence Service could potentially crash if a Durable Subscription name was longer than 512 characters. This was due to the absence of bounds checking when performing an internal copy operation.

[RTI Issue ID PERSISTENCE-337]

5.1.2 [Critical] Crash in Persistence Service when using ContentFilteredTopics and sharing type codes for discovery

Persistence Service could crash if the configuration used ContentFilteredTopics and type information was shared through discovery via Type Codes.

[RTI Issue ID PERSISTENCE-332]

5.2 Memory Leaks/Growth

5.2.1 [Major] Persistence Service could leak a file handle on Windows when purging sample logs upon startup

Persistence Service had a resource leak on Windows where a file handle wasn't closed when purging the sample logs (see Sample Logging Tags in the *RTI Connext Core Libraries User's Manual*) for a Persistence Group upon startup. Sample log files from a previous execution are typically purged when starting the *Persistence Service*.

[RTI Issue ID PERSISTENCE-336]

5.3 Other

5.3.1 [Critical] Configuring persistence group's DataWriter QoS to use compression and XCDR2 encapsulation caused samples to not be received from Persistence Service

QoS of a persistence group did not work. For example:

```
<persistence_group name="HelloWorldGroup">
  <filter>*</filter>
  <datawriter_qos>
    <representation>
      <value>
        <element>XCDR2_DATA_REPRESENTATION</element>
      </value>
      <compression_settings>
        <compression_ids>LZ4</compression_ids>
      </compression_settings>
    </representation>
  </datawriter_qos>
</persistence_group>
```

Using the above configuration could have caused *DataReaders* to not receive samples from the *DataWriter* associated with the persistence group.

For *DataReaders* using a ContentFilteredTopic, you may have seen the following error in *Persistence Service*:

```
DDS_SqlFilter_evaluateOnSerialized:deserialization error: sample"
```

For *DataReaders not* using a ContentFilteredTopic, you may have seen the following deserialization error in your applications:

```
PRESCstReaderCollator_storeSampleData:deserialize sample error in topic
↪'MyTopic' with type 'MyType'
```

For a *Routing Service DataReader* receiving samples from *Persistence Service*, you might have seen the following error:

```
DDS_DynamicData2_from_cdr_buffer:deserialization error: sample
```

[RTI Issue ID PERSISTENCE-318]

5.3.2 [Critical] <content_filter> tag caused segmentation fault in Persistence Service if filter expression was invalid

Using the `<content_filter>` tag in *Persistence Service*, which applies Content Filters on the incoming *DataWriter* samples, crashed if the filter expression was invalid. The expression could have been invalid if:

1. The published topic's samples didn't contain the fields specified in the value of `<content_filter>`.
2. The filter expression specified in the `<content_filter>` was syntactically incorrect based on the Queries and Filters Syntax section in the API Reference.

[RTI Issue ID PERSISTENCE-281]

5.3.3 [Critical] DataReader stopped receiving samples from Persistence Service DataWriter that uses DDS fragmentation when Persistence Service ran in PERSISTENT mode

A *DataReader* may have stopped receiving samples from a *Persistence Service DataWriter* configured to use DDS fragmentation (asynchronous publishing with samples that exceed the minimum **mesage_size_max** across all installed transports) when *Persistence Service* ran in PERSISTENT mode (the *DataWriter* stores the samples on disk). The issue occurred when a sample fragment was lost, which is more likely to occur in lossy networks.

[RTI Issue ID PERSISTENCE-323]

5.3.4 [Critical] Error when receiving compressed batch samples

Persistence Service failed to deserialize compressed batch samples. You would have seen errors like these:

```
[0x01013446,0xCA403602,0xE81D6C90:0x80000107{E=DR,
I=24}|RECEIVE FROM 0x01019DBE,0xD54C0C6F,0x1A7264AB:0x80000002]
PRESCstReaderCollator_storeSampleData:!update stream off-
set [0x01013446,0xCA403602,0xE81D6C90:0x80000107{E=DR,
I=24}|RECEIVE FROM 0x01019DBE,0xD54C0C6F,0x1A7264AB:0x80000002]
PRESCstReaderCollator_storeSampleToEntry:!store sample
data [0x01013446,0xCA403602,0xE81D6C90:0x80000107{E=DR,
I=24}|RECEIVE FROM 0x01019DBE,0xD54C0C6F,0x1A7264AB:0x80000002]
PRESCstReaderCollator_newData:!get entries [0x01013446,0xCA403602,
0xE81D6C90:0x80000107{E=DR,I=24}|RECEIVE FROM 0x01019DBE,0xD54C0C6F,
0x1A7264AB:0x80000002] PRESCstReaderCollator_storeSampleData:!update
stream offset [0x01013446,0xCA403602,0xE81D6C90:0x80000107{E=DR,
I=24}|RECEIVE FROM 0x01019DBE,0xD54C0C6F,0x1A7264AB:0x80000002]
PRESCstReaderCollator_storeSampleToEntry:!store sample
data [0x01013446,0xCA403602,0xE81D6C90:0x80000107{E=DR,
I=24}|RECEIVE FROM 0x01019DBE,0xD54C0C6F,0x1A7264AB:0x80000002]
PRESCstReaderCollator_newData:!get entries
```

[RTI Issue ID PERSISTENCE-321]

5.3.5 [Minor] <configuration_variables> tag not supported in Persistence Service

The `<configuration_variables>` tag was visible and accepted by the *Persistence Service* .xsd file, but it had no effect: the parsing of an XML file with this tag to set the value of XML-defined environment variables failed. This problem has been corrected. Now, the `<configuration_variables>` tag can be used to define default values for XML-defined environment variables, which will take effect if those environment variables are not set on the terminal.

For example:

```
<dds>
    <configuration_variables>
        <value>
            <element>
                <name>STORAGE_LOCATION</name>
                <value>/tmp/persistence_service_storage</value>
            </element>
        </value>
    </configuration_variables>
    <persistence_service name="defaultDisk">
        <persistent_storage>
            <filesystem>
                <directory>$(STORAGE_LOCATION)</directory>
            </filesystem>
        </persistent_storage>
        <participant name="defaultParticipant">
            <persistence_group name="persistAll">
                <filter>*</filter>
                <single_publisher>true</single_publisher>
                <single_subscriber>true</single_subscriber>
                <datawriter_qos base_name="SystemPersistenceQosLib::QosProfile
↪"/>
                <datareader_qos base_name="SystemPersistenceQosLib::QosProfile
↪"/>
            </persistence_group>
        </participant>
    </persistence_service>
</dds>
```

[RTI Issue ID PERSISTENCE-306]

5.3.6 [Minor] Memory Leak when parsing environment variable in XML that does not exist

When parsing an environment variable in XML, if the environment variable does not exist, *Connext* logs an error and terminates parsing of the XML containing the error. The code handling this error condition did not properly free memory allocated during the parsing of the XML up to the point of the error, resulting in a memory leak.

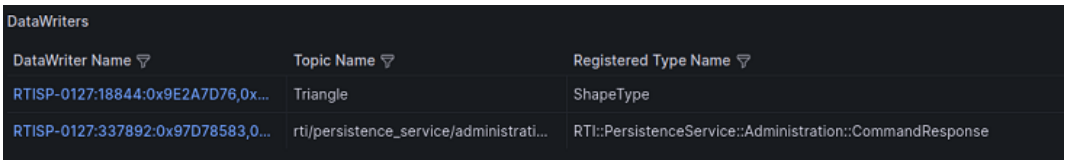
[RTI Issue ID PERSISTENCE-309]

Chapter 6 Previous Releases

6.1 What's New in 7.2.0

6.1.1 Persistence Service compatible with Monitoring Library 2.0

RTI Monitoring Library 2.0 can now be enabled in *Persistence Service* so that all DDS entities created by this service will provide monitoring data to *RTI Observability Framework*.



To enable *Monitoring Library 2.0* in *Persistence Service*, add the XML code snippet shown below to an XML QoS profile, then run *Persistence Service* from the folder containing the profile. Add the snippet to any of the following XML files:

- `NDDS_QOS_PROFILES.xml`, located in the *Persistence Service* working directory
- `USER_QOS_PROFILES.xml`, located in the *Persistence Service* working directory
- Any XML file included in the `NDDS_QOS_PROFILES` environment variable

```
<?xml version="1.0"?>
<dds>
    <qos_library name="MonitoringEnabledLibrary">
        <qos_profile name="MonitoringEnabledProfile"
                     is_default_participant_factory_profile="true">
            <participant_factory_qos>
                <monitoring>
                    <enable>true</enable>
                </monitoring>
            </participant_factory_qos>
        </qos_profile>
    </qos_library>
</dds>
```

See also:

- How to Load XML-Specified QoS Settings, in the *RTI Connext Core Libraries User's Manual*

- Monitoring Library 2.0

6.1.2 Support for dynamic certificate revocation and renewal without needing to restart Persistence Service

A running Persistence Service instance can use the new `authentication.crl_file_poll_period.millisec` and `authentication.identity_certificate_file_poll_period.millisec` properties in the SECURITY PLUGINS *(RTI Security Plugins)* to specify certificate revocations and renewals without the need to restart the service. (In release 7.3, these two properties are replaced by a single property, `files_poll_interval`.) These properties must have a value greater than zero for the participant to periodically poll the provided security-related files for changes.

For more information, see Advanced Authentication Concepts in the *RTI Security Plugins User's Manual*.

6.1.3 New Library API functions in C allow get/set QoS operations on internal DomainParticipants

Persistence Service supports a new set of Library API functions in C that allow updating the QoS values *DomainParticipants* use to receive and publish data for the Persistence Groups contained in participants (<participant> is the corresponding tag in the XML configuration). These new APIs can provide additional flexibility in use cases that require changing a *DomainParticipant* QoS. For example, you could use them to update:

- QoS properties
- Transport configuration
- Participant partitions

In addition, you can use these APIs to change the configuration of the two *DomainParticipants* associated with a <participant> tag in XML separately. This option was not possible using XML.

The new Library API names are:

- `RTI_PersistenceService_get_participant_qos`
- `RTI_PersistenceService_set_participant_qos`

For more details, see the Persistence Service API Reference.

6.1.4 New Library API functions in C allow get/set QoS operations on internal Publisher/Subscriber entities

Persistence Service supports a new set of Library API functions in C that allow updating the QoS values of the *Publisher/Subscriber* entities belonging to Persistence Groups (<persistence_group> is the corresponding tag in the XML configuration). These new APIs can provide additional flexibility in use cases that require changing the *Publisher/Subscriber* QoS. For example, you could use them to update:

- QoS properties
- Endpoint partitions

The new Library API names are:

- `RTI_PersistenceService_get_publisher_qos`
- `RTI_PersistenceService_set_publisher_qos`
- `RTI_PersistenceService_get_subscriber_qos`
- `RTI_PersistenceService_set_subscriber_qos`

For more details, see the Persistence Service API Reference.

6.1.5 Persistence Service now allows you to set rtps_app_id, giving you greater control over value of DomainParticipant's GUID prefix

Persistence Service now allows you to set the value for `rtps_app_id` in a *DomainParticipant's* WIRE PROTOCOL QoS policy. In previous releases, if you tried to set this value, *Persistence Service* would log an exception and override the value with `DDS_RTPS_AUTO_ID`.

This new capability allows you to have greater control over the value of a *DomainParticipant's* GUID prefix. It can especially be helpful if you want to use your own `rtps_app_id` to identify the *DomainParticipants* created by an instance of *Persistence Service* by looking at the GUID prefix encapsulated as part of the RTPS wire packet's header. For example, a layer 7 load balancer could make routing decisions based on the value of the `rtps_app_id` of the *DomainParticipant's* GUID prefixes.

Note that *Persistence Service* still ensures that the GUID prefixes for all its *DomainParticipants* are unique, even if they share the same `rtps_app_id`, by automatically generating the `rtps_instance_id`, which is the last part of the GUID prefix. This last part is not user-configurable.

6.2 What's Fixed in 7.2.0

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

6.2.1 [Critical] Segmentation fault issue with Persistence Service and Distributed Logger

Previous releases encountered a segmentation fault when using the *Persistence Service* library. This issue occurred when attempting to use the Distributed Logger after deleting an `RTI_PersistenceService` object. The problem occurred because the call to the `RTI_PersistenceService_delete` operation was deleting the Distributed Logger instance.

This problem is now fixed. If the user application creates the Distributed Logger instance instead of relying on its creation through the <distributed_logger> XML tag by *Persistence Service*, the deletion of the *Persistence Service* instance will no longer attempt to delete it.

[RTI Issue ID PERSISTENCE-297]

6.2.2 [Major] Error creating a persistence group DataWriter when setting dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size to -1

Creating a persistence group *DataWriter* could fail with the following error if the property `dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size` was set to -1 in the <datawriter_qos>:

```
"!allocate sample buffer pool"
```

Even if the *DataWriter* creation did not fail, the value of `dds.data_writer.history.odbc_plugin.builtin.sample_cache_max_size` would be incorrectly applied. Instead, it would be set to `dds.data_writer.history.odbc_plugin.builtin.instance_cache_max_size` for keyed topics and 1 for unkeyed topics.

This problem has been resolved.

[RTI Issue ID PERSISTENCE-296]

6.2.3 [Minor] <reader_checkpoint_frequency> may not have been applied correctly

There was an issue applying the configuration parameter <reader_checkpoint_frequency> when the value was different than 1. The desired frequency for storing the `PRSTDataReader` state might not have been accurately applied, resulting in a delayed storage process. This problem has been resolved.

[RTI Issue ID PERSISTENCE-307]

6.3 What's New in 7.1.0

6.3.1 Persistence Service support as a library for all supported architectures

This release adds support for *Persistence Service* Library API (static and dynamic), as an alternative to the standalone executable available in previous releases. Now you can run a *Persistence Service* instance within your application by linking with the new library and using the C API offered by the library on all supported architectures. Previously this support was available only for INTEGRITY® systems.

For additional information on the *Persistence Service* Library API, see the Persistence Service API Reference.

For an example of how to use *Persistence Service* as a library, see the rticonnextdds-examples section on GitHub.

6.3.2 Removed ability to share a database connection in Persistence Service and durable writer history

This release removes the ability to share a database connection in *RTI Persistence Service* (which is done by setting the tag <share_database_connection> to true for a <persistence_group>). It also removes the ability to share a database connection when using durable writer history and setting the property `dds.data_writer.history.odbc_plugin.builtin.shared` to 1.

Note that sharing a database connection was only allowed for external databases, and support for external databases was removed in 7.0.0.

6.3.3 Third-party software upgrade

The following third-party software used by *Persistence Service* has been upgraded:

Third-Party Software	Previous Version	Current Version
SQLite®	3.39.0	3.39.4

For information on third-party software used by *Connext* products, see the "3rdPartySoftware" documents in your installation: **<NDDSHOME>/doc/manuals/connext_dds_professional/release_notes_3rdparty**.

6.4 What's Fixed in 7.1.0

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

6.4.1 [Major] Persistence Service stored/forwarded samples multiple times when there were two or more equivalent versions of a type for a Topic

Persistence Service stored and forwarded incoming samples multiple times when there were two or more equivalent versions of a type for a given *Topic* in the system.

Two types are equivalent when they only differ on typedef. For example, MyType and MyType2 are equivalent in this IDL snippet:

```
struct MyType {
    long m1;
};

typedef long MyLong;

struct MyType2 {
    MyLong m1;
};
```

This problem has been fixed.

[RTI Issue ID PERSISTENCE-269]

6.4.2 [Major] Unexpected fatal error when number of instances reached the limit *

In 7.0.0, an unexpected fatal error could be logged when the following occurred:

- *Persistence Service* was running in PERSISTENT mode.
- The number of instances reached the `max_instances` limit set in one of the *Persistence Service DataWriters'* RESOURCE_LIMITS QoS.
- *Connext* could not find an instance to delete (such as an unregistered one), to replace with the new instance. So the new instance could not be added.

This log message is expected, but it is not a fatal error, so its verbosity has been updated to WARNING, as follows:

```
WriterHistoryOdbcPlugin_createResources:FIND FAILURE | Instance for␣
↪replacement
```

```
WriterHistoryOdbcPlugin_createResources:FIND FAILURE | Instance for␣
↪replacement
WriterHistoryOdbcPlugin_addInstance:OUT OF RESOURCES | Exceeded the number of␣
↪instances. Current registered instances (128), maximum number of instances␣
↪(128)(writer_qos.resource_limits.max_instances)
```

[RTI Issue ID CORE-13496]

6.4.3 [Minor] Persistence Service XSD schema was broken *

In release 7.0.0, the *Persistence Service* XSD schema was broken due to an additional closing tag. This was a regression that only affected the 7.0.0 release. This issue has been fixed.

[RTI Issue ID PERSISTENCE-276]

6.4.4 Fixes related to vulnerabilities

[Critical] Potential arbitrary SQL query execution when enabling database locking

There was the potential for arbitrary SQL query execution in *Persistence Service* running with database locking enabled (which is not the default setting). This issue has been fixed.

User Impact without Security

A SQL Injection vulnerability in *Persistence Service* could have resulted in the following:

- Arbitrary SQL query execution.
- Exploitable from the same host Persistence Service is running.
- Potential impact on integrity and confidentiality of Persistence Service.
- CVSS Base Score: 7.1 HIGH
- CVSS v3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N

User Impact with Security

Same impact as described for "User Impact without Security" above.

[RTI Issue ID PERSISTENCE-272]

* *This bug does not affect you if you are upgrading from 6.1.x or earlier.*

6.5 What's New in 7.0.0

6.5.1 Support for external databases is discontinued

External databases are no longer supported by Persistence Service.

The Release Notes for 6.1.1 included a deprecation notice, in keeping with the Real-Time Innovations, Inc. Maintenance Policy #4220.

6.5.2 Default journal_mode and synchronization changed to WAL and NORMAL, respectively

In this release, the default values for the following configuration parameters have changed:

- <journal_mode> has changed from DELETE to WAL
- <synchronization> has changed from OFF to NORMAL

This change provides the best out-of-the-box performance without sacrificing database integrity in the event of a crash or power failure.

6.5.3 Third-party software upgrade

The following third-party software used by *Persistence Service* has been upgraded:

Third-Party Software	Previous Version	Current Version
SQLite®	3.37.2	3.39.0

For information on third-party software used by *Connext* products, see the "3rdPartySoftware" documents in your installation: **<NDDSHOME>/doc/manuals/connext_dds_professional/release_notes_3rdparty**.

6.6 What's Fixed in 7.0.0

[Critical]: System-stopping issue, such as a crash or data loss. [Major]: Significant issue with no easy workaround. [Minor]: Issue that usually has a workaround. [Trivial]: Small issue, such as a typo in a log.

6.6.1 [Critical] Fatal error when persisting unkeyed Topics upon restore or IP mobility event

Persistence Service generated the following fatal error and shut down when persisting unkeyed Topics if *all* of the following conditions were met:

- <use_durability_service> was set to true in the <persistence_group> OR <writer_qos>/<writer_data_lifecycle>/<autopurge_disposed_instances_delay> was set to zero in the <persistence_group>
- <writer_in_memory_state> was set to false in the <persistence_group>.
- There was an IP mobility event (for instance, an interface went down) OR *Persistence Service* was started with the `-restore` command-line option set to true.

The error backtrace was as follows:

```
#4 WriterHistoryOdbcPlugin_logAndCheckODBCError ??:? [0x31590B]
#5   WriterHistoryOdbcPlugin_handleODBCError ??:? [0x315CE5]
#6   WriterHistoryOdbcPlugin_beginDisposedInstanceIteration ??:? [0x34B202]
```

This problem has been resolved.

[RTI Issue ID PERSISTENCE-255]

6.6.2 [Major] Samples published out of order from same virtual GUID were dropped

If *Persistence Service* received samples for a given virtual GUID with sequence numbers out of order, *Persistence Service* dropped samples with sequence numbers lower than the highest received sequence number. This issue has been resolved.

[RTI Issue ID PERSISTENCE-250]

6.6.3 [Minor] Schema files not compliant with DDS-XML specification

The following change has been made to the schema file `rti_persistence_service.xsd`, and its included files, to make them compliant with the DDS-XML specification (https://www.omg.org/spec/DDS-XML/1.0/PDF):

- Renamed <participant_qos> to <domain_participant_qos>

The old tag is still accepted by the *Connext* XML parser and the XSD schema to maintain backward compatibility.

[RTI Issue ID PERSISTENCE-213]

Chapter 7 Known Issues

Note: For an updated list of critical known issues, see the Critical Issues List on the RTI Customer Portal at https://support.rti.com.

7.1 Coherent Changes not Propagated as Coherent Set

Persistence Service will propagate the samples inside a coherent change. However, it will propagate these samples individually, not as a coherent set.

7.2 TopicQueries not Supported in PERSISTENT Mode

Getting TopicQuery data from a Persistence Service instance configured to store data on disk is not currently supported.

Note: Getting TopicQuery data from a Persistence Service instance running in TRANSIENT (storing data in memory) mode is supported.

[RTI Issue ID PERSISTENCE-143]

7.3 <comm_ports> not Supported when Using Real-Time WAN Transport

Persistence Service can use the *RTI Real-Time WAN Transport*. However, the port configuration using <comm_ports> or the property `dds.transport.UDPv4_WAN.builtin.comm_ports` is not currently supported by *Persistence Service*.

[RTI Issue ID PERSISTENCE-206]

7.4 Persistence Service DataReaders Ignore Serialized Key Propagated with Dispose Updates

Persistence Service DataReaders ignore the serialized key propagated with dispose updates. *Persistence Service DataWriters* cannot propagate the serialized key with dispose, and therefore ignore the `serialize_key_with_dispose` setting on the *DataWriter* QoS.

[RTI Issue ID PERSISTENCE-221]

7.5 Synchronizing Data Samples between Persistence Service Instances Can Consume Significant Bandwidth

Synchronizing data samples between *Persistence Service* instances (using `<synchronize_data>1</synchronize_data>`; see Synchronizing of Persistence Service Instances in the *Core Libraries User's Manual*) consumes a significant amount of bandwidth. This is because a *Persistence Service* instance does not keep track of which other *Persistence Service* instances have already received the data, causing unnecessary traffic.

For example:

1. Start Persistence Service 1 (PS1) with synchronization enabled.
2. Publish samples to PS1.
3. Start PS2 with synchronization enabled.
4. PS1 will send samples to PS2 since synchronization is enabled on both.
5. **PS2 will also send samples back to PS1 because it doesn't know PS1 already received the samples.** Each data sample will be sent N * (N - 1) times on the network for data synchronization to fully complete.

[RTI Issue ID PERSISTENCE-266]

7.6 Some tags in the XML configuration must be grouped in a strict order

The XML validator tools *Persistence Service* uses to validate XML configuration files adhere to the XML 1.0 specification, which doesn't offer a way of defining collections of unordered tags that are both bounded and unbounded in occurrences.

This limitation is no longer present in XML 1.1. However, there are no C or C++ validators compliant with the XML 1.1 specification at the time of writing.

[RTI Issue ID CORE-14178]

Chapter 8 Available Documentation

The following documentation is provided with the *Persistence Service* distribution. (The paths show where the files are located after *Persistence Service* has been installed in **<NDDSHOME>**):

- General information, configuration, use cases, and execution of *Persistence Service* can be found in the RTI Connext Core Libraries User's Manual (**<NDDSHOME>/doc/manuals/connext_dds_professional/users_manual/RTI_ConnextDDS_CoreLibraries_Us**
- By default, *Persistence Service* example code is copied here:
  - macOS systems: **/Users/<your user name>/rti_workspace/version/examples/persistence_service/<language>/hello_world_persistence**
  - Linux systems: **/home/<your user name>/rti_workspace/version/examples/persistence_service/<language>/hello_world_persistence**
  - Windows systems: **<your home directory>rti_workspaceversionexamplespersistence_service<language>/hello_world_persistence**
- Overview of persistence and durability features: Open **<NDDSHOME>/README.html**, choose your desired API (C, C++, or Java), then select **Modules**, **RTI Connext API Reference**, **Durability and Persistence**.