RTI Connext Core Libraries Release Notes

Version 7.7.0

Contents

1 System Requirements **1**
- 1.1 Supported Platforms . . . 2
- 1.2 Disk and Memory Usage . . . 3

2 Compatibility **4**
- 2.1 Wire Protocol Compatibility . . . 4
- 2.2 Code and Configuration Compatibility . . . 5
- 2.3 Extensible Types Compatibility . . . 5

3 What's New in 7.7.0 **6**
- 3.1 Discovery . . . 6
  - 3.1.1 Reduce network traffic and system startup time by sending full type information only when needed . . . 6
  - 3.1.2 type_object_max_serialized_length defaults to LENGTH_AUTO, in support of new TypeObject v2 default . . . 7
  - 3.1.3 LBED Plugin now supports setting DataRepresentationQosPolicy . . . 8
- 3.2 Performance and Scalability . . . 8
  - 3.2.1 Significant memory footprint reduction for Connext applications . . . 8
  - 3.2.2 Reduced default metrics set in Monitoring Library 2.0, minimizing bandwidth utilization . . . 9
  - 3.2.3 Improved discovery and reduced overhead with Collector Service . . . 9
  - 3.2.4 Improved write execution time when dds.data_writer.history.purge_samples_after_acknowledgment set to 1 . . . 10
  - 3.2.5 Durable Writer History performance improvements when application-level acknowledgments enabled . . . 10
  - 3.2.6 Priority inheritance used when creating semaphores in Linux . . . 11
  - 3.2.7 Improved responsiveness to remote reader configuration updates . . . 11
  - 3.2.8 Performance improvement loading XML QoS profiles . . . 11
  - 3.2.9 Improved performance of Dynamic Data-to-text format functions . . . 11
  - 3.2.10 DomainParticipants rejoin the system faster . . . 11
  - 3.2.11 Disable reliable writer communication with best effort reader to improve performance and memory usage . . . 11
  - 3.2.12 Improve performance of durable DataWriters when sending data to late-joining DataReaders, using new property . . . 12
- 3.3 APIs . . . 12
  - 3.3.1 New resource limit gives more control over memory allocation . . . 12
  - 3.3.2 New APIs allow your application code to check for type assignability issues . . . 12

3.3.3 Option to lazily allocate batching resources in DataReaders, reducing unnecessary memory usage . . . 13
3.3.4 Python API supports versions 3.10 to 3.14, no longer supports versions 3.8 and 3.9 . 13
3.4 Usability . . . 14
3.4.1 Monotonic clock as new default enhances robustness against system clock changes . . 14
3.4.2 Increased maximum length of command-line parameters for rtiddsspy . . . 14
3.4.3 Built-in QoS Snippets added to easily enable or disable TypeObject v1 and TypeObject v2 . . . 14
3.4.4 Flexible control over Monitoring Library 2.0 enablement using new environment variable . . . 14
3.5 Debuggability . . . 15
3.5.1 Monitoring Library 2.0 now enabled by default under specific conditions, making it easier to use remote debugging . . . 15
3.5.2 Monitoring Library 2.0 security enforced when using an RTI security plugin, elevating security standards . . . 15
3.5.3 Improved diagnostics through extended SQLite error codes in durable writer and reader state databases . . . 16
3.5.4 Debug symbols for release libraries now provided for Linux platforms . . . 16
3.6 OMG Specification Compliance . . . 16
3.6.1 Better control communication in a system with changing types and evolving data model requirements, using @must_understand annotation . . . 16
3.6.2 Augment your data models by explicitly describing units of measurement associated with types . . . 16
3.7 DDS Spy . . . 17
3.7.1 Improved transport configuration in DDS Spy with alias-based option . . . 17
3.7.2 Support for Real-Time WAN Transport in DDS Spy . . . 17
3.7.3 Support for participant discovery events in DDS Spy . . . 17
3.7.4 DDS Spy option for printing Publisher and Subscriber Object IDs for easier diagnostics 18
3.7.5 New DDS Spy option for QoS inspection to help identify incompatibilities that prevent communication . . . 18
3.7.6 New DDS Spy options for type inspection . . . 19
3.7.7 DDS Spy enhanced source locator display with IPv6 and shared memory support . . 19
3.8 Deprecations and Removals . . . 19
3.8.1 Python 3.8 and 3.9 no longer supported . . . 20
3.9 Third-Party Software Changes . . . 20
3.10 Other . . . 20
3.10.1 Improved isolation for observability traffic . . . 20
3.10.2 Aliases for enabled_transport QoS no longer require case-sensitive matching . . . 21
3.10.3 Disallow unrealistic values of polling_period, to avoid errors collecting data for Observability Framework . . . 21
3.10.4 Allow disabling signal-blocking on Connext Threads for POSIX platforms . . . 21
3.10.5 Convert DynamicType (or TypeCode) to JSON schema, for enhanced interoperability 21
3.10.6 Ensure compatibility with other DDS vendors using updated Compatibility.OtherDDSVendor.Enable QoS snippet . . . 22
3.10.7 Added operation to retrieve remaining time until license expiration . . . 22

4 What's Fixed in 7.7.0 **23**
4.1 Serialization and Deserialization . . . 23

4.1.1 [Critical] Endpoint serialization/deserialization failure after deleting DomainParticipants if setting different xTypes compliance masks with NDDS_Config_set_xtypes_compliance_mask . . . 23
4.2 Usability . . . 24
4.2.1 [Minor] Lack of allowed multicast interfaces did not cause Connext to ignore multicast addresses . . . 24
4.2.2 [Minor] Error when configuring AllocationSettings with initial equals to maximum . 24
4.2.3 [Minor] Monitoring Library 2.0 required a license when used in DDS Ping or DDS Spy with license-managed bundles . . . 25
4.3 Transports . . . 25
4.3.1 [Critical] Applications using Real-Time WAN may have stopped communicating after IP mobility events . . . 25
4.3.2 [Major] IP mobility functionality did not detect when network interfaces gained multicast capability . . . 26
4.3.3 [Minor] Zero-Copy DataReaders did not filter out redundant samples . . . 26
4.3.4 [Minor] Incorrect information in shared memory log message when out of resources 26
4.3.5 [Minor] DomainParticipant failed to be enabled if a network interface with UDPv4 address 0.0.0.0 was detected . . . 26
4.4 Reliability Protocol and Wire Representation . . . 27
4.4.1 [Critical] Sample may not have been delivered to a DataReader in a Required Subscription . . . 27
4.4.2 [Critical] Samples lost if new DataReader created on same topic and DomainParticipant as existing DataReaders using async publishing . . . 27
4.4.3 [Critical] Sample may not have been delivered to a DataReader using application-level acknowledgments . . . 27
4.4.4 [Major] Unexpected precondition error sending repair to DataReaders with 16 locators 28
4.5 Performance and Scalability . . . 28
4.5.1 [Critical] Performance degradation with durable writer history . . . 28
4.5.2 [Critical] Performance degradation with Durable Writer History in write-intensive scenarios . . . 28
4.5.3 [Major] Inefficient repair of fragmented samples caused performance degradation during large data transmission . . . 29
4.6 Debuggability . . . 29
4.6.1 [Minor] polling_period was not required to be a positive value . . . 29
4.6.2 [Minor] take_discovery_snapshot may have provided inaccurate information for remote endpoints . . . 29
4.6.3 [Minor] Incorrect timestamp in log messages with microseconds higher than 500000 29
4.6.4 [Trivial] Backtraces did not correctly print addresses higher than 0xFFFFFFFF . . . 30
4.7 TopicQueries . . . 30
4.7.1 [Major] Inefficient O(n^2) reader-side performance when creating multiple TopicQueries simultaneously . . . 30
4.8 Logging . . . 30
4.8.1 [Trivial] Improved log messages for inconsistent WireProtocolQosPolicy values . . . 30
4.8.2 [Trivial] Confusing log message when DataWriter could not queue items * . . . 31
4.9 APIs (C or Traditional C++) . . . 31
4.9.1 [Minor] DDS_DataWriter_is_sample_app_acknowledged didn't check for NULL for is_app_ack parameter . . . 31
4.10 APIs (Modern C++ API) . . . 31

4.10.1 [Critical] Implicit destruction of Modern C++ entities inside their Listeners 31
4.10.2 [Major] Excessive memory allocation in rti::core::xtypes::convert function 31
4.10.3 [Major] find_datawriters/readers APIs didn't work with XML-Based Application Creation 32
4.10.4 [Minor] Incorrect implementation of BuiltinTopicReaderResourceLimits default constructor 32
4.10.5 [Minor] Missing Modern C++ Distributed Logger import library on Windows 33
4.11 APIs (Java) 33
4.11.1 [Major] Some BuiltinTopicKey_t's functions did not work properly 33
4.12 APIs (C# API) 33
4.12.1 [Major] .NET API built with only .NET Standard 2.0 as target framework 33
4.12.2 [Minor] DomainParticipant creation may not have respected EntityFactory auto-enable policy in C# API 34
4.13 APIs (Python) 34
4.13.1 [Trivial] Python API: mypy failures on Windows due to non-ASCII character in doc-strings 34
4.14 Dynamic Data 34
4.14.1 [Minor] Unbounded Sequences incorrectly converted to string from DynamicType . 34
4.15 XML Configuration 35
4.15.1 [Minor] Setting monitoring.telemetry_data.metrics in a derived profile or using an XML snippet did not override inherited values 35
4.16 Entities 37
4.16.1 [Critical] TransportMulticast QoS locators not properly tracked by locator reachability * 37
4.16.2 [Minor] Misleading log messages when printing or converting to string non-default TransportMulticast QoS policy 37
4.17 Crashes 37
4.17.1 [Critical] Possible crash if configuration of Monitoring Library 2.0 was remotely modified while an entity was being destroyed 37
4.17.2 [Critical] Crash when dividing by zero in XML type definition 38
4.17.3 [Critical] DataWriter using Durable Writer History and not using Required Subscription crashed when communicating with a DataReader with a role_name 38
4.17.4 [Critical] Possible crash when deleting an Entity while Monitoring Library 2.0 was enabled 38
4.17.5 [Critical] Crash when concurrently modifying QoS profiles with Monitoring QoS policy set in default factory profile 39
4.17.6 [Critical] Crash when using DDS_RECOVER_INSTANCE_STATE_CONSISTENCY and multiple writers updating the same instances 39
4.17.7 [Critical] Segmentation fault in DataReader that was created disabled when using instance state consistency 39
4.17.8 [Major] Precondition added to rti::flat::StringBuilder::set_string() and rti::flat::PrimitiveSequenceBuilder<T>::add_n() 40
4.18 Hangs 40
4.18.1 [Critical] Possible hang in TCP transport when failing to send a message 40
4.19 Memory Leaks/Growth 40
4.19.1 [Critical] Memory use by Monitoring Library 2.0 may have grown unbounded over time 40

4.19.2 [Critical] Unbounded memory growth when setting receive_window_size on DataReader . . . 40
4.19.3 [Critical] Memory leak in DataWriter when matching reliable DataReader . . . 41
4.19.4 [Critical] History depth not honored after instance transitioned from NOT_ALIVE to ALIVE state . . . 41
4.19.5 [Major] Memory leak when UDP receive socket creation failed . . . 41
4.19.6 [Major] Memory leak in Monitoring Library 2.0 when mixing library types . . . 42
4.19.7 [Minor] Unbounded memory growth in Modern C++ when creating and deleting DynamicData topics of the same type . . . 42
4.19.8 [Minor] Memory leak when disabling Monitoring Library 2.0 in Traditional C++ . . . 42
4.20 Data Corruption . . . 43
4.20.1 [Critical] Data corruption in Monitoring Library 2.0 due to race condition during event and log processing . . . 43
4.20.2 [Major] Monitoring Library 2.0 failed to create dependent Entities when type_object_max_serialized_length and type_code_max_serialized_length were 0 * . . . 43
4.20.3 [Minor] Converting a GUID_t to a BuitinTopicKey_t had issues when high bit was set 43
4.21 OMG Specification Compliance . . . 44
4.21.1 [Major] Samples of mutable types may have been dropped when bit FLAG_MUST_UNDERSTAND was set on key members . . . 44
4.22 Interoperability . . . 44
4.22.1 [Major] RTPS Header Extension timestamp used incorrect clock . . . 44
4.22.2 [Major] TypeConsistencyEnforcement::ignore_member_names did not work in some cases . . . 45
4.23 Vulnerabilities . . . 46
4.23.1 [Critical] Potential unauthorized access to instance information in Connext applications 46
4.23.2 [Critical] Potential Denial of Service in Connext applications when parsing XML files 47
4.23.3 [Critical] Potential heap buffer write overflow and memory leak in Connext applications when parsing an XML type . . . 48
4.23.4 [Critical] Potential heap buffer read overflow in Connext applications when parsing an XML type . . . 49
4.23.5 [Critical] Potential out-of-bounds write in Connext when using Durable Writer History 49
4.23.6 [Critical] Potential external entity attack (XXE) in Connext applications when parsing malicious XML files . . . 50
4.23.7 [Critical] Potential heap-based buffer overflow when parsing malicious XML configuration files . . . 51
4.23.8 [Critical] Potential out-of-bounds read in Core Libraries on invalid XML constant expression . . . 52
4.24 Other . . . 53
4.24.1 [Critical] AppAck messages exceeded underlying transport message size . . . 53
4.24.2 [Trivial] Incorrect partition labels in DDS Spy . . . 53

5 Previous Releases **54**
5.1 What's New in 7.6.0 . . . 54
5.1.1 Updated default heartbeat periods for faster repairs of lost samples . . . 54
5.1.2 DNS tracker now enabled by default to reduce chance of discovery problems when using hostnames in peer lists . . . 55
5.1.3 Extensible Types compliance mask added to all APIs . . . 55
5.1.4 Updated code generation for 8-bit integer types . . . 55

5.1.5 Application no longer fails if platform does not support SOCKET_BUFFER_SIZE_OS_MAX . . . 55
5.1.6 Improved throughput when disable_positive_acks field in DataReaderProtocol QoS policy set to TRUE . . . 56
5.1.7 New metadata fields for custom filters in modern C++, Java, and Python . . . 56
5.1.8 TCP Transport now uses POLL socket monitoring by default on Linux systems, offering significantly better responsiveness . . . 56
5.1.9 Decreased bandwidth usage when not actively debugging remote systems . . . 56
5.1.10 New built-in profile makes it easier to monitor Connext applications across geographically separated networks . . . 57
5.1.11 Third-Party Software Changes . . . 57
5.2 What's Fixed in 7.6.0 . . . 58
5.2.1 Discovery . . . 58
[Critical] Port collision on one channel of multichannel DataWriter caused discovery to fail even if another channel had no port collision . . . 58
5.2.2 Serialization and Deserialization . . . 59
[Critical] Unexpected error messages when disposing of an instance while using batching and setting serialize_key_with_dispose to TRUE . . . 59
5.2.3 Usability . . . 59
[Critical] Locator Reachability DataReader could not receive assertions from more than four DomainParticipants . . . 59
[Minor] DDS_RTPS_AUTO_ID_FROM_MAC failed if MAC address had all zeros for first 4 bytes or last 4 bytes . . . 60
[Minor] Inconsistent QoS error when enabling Monitoring Library 2.0 with a default profile (is_default_qos = TRUE) . . . 60
5.2.4 Transports . . . 60
[Major] TCP Transport did not properly enforce system FD_SETSIZE hard limits for select() socket monitoring . . . 60
[Minor] Possible failure to create DomainParticipant if using multicast . . . 61
5.2.5 Reliability Protocol and Wire Representation . . . 61
[Critical] Unexpected data loss when using batching and finite reader resource limits . 61
5.2.6 Performance and Scalability . . . 61
[Critical] Slower DDS Entity creation in large applications . . . 61
5.2.7 Debuggability . . . 62
[Major] Duplicate network traffic captures when creating WaitSets, AsyncWaitSets, or GuardConditions . . . 62
5.2.8 Content Filters and Query Conditions . . . 62
[Critical] Unexpected data loss when using writer-side content filtering and changing QoS . . . 62
[Critical] Historical sample loss for late-joining DataReaders setting protocol.virtual_guid . . . 63
5.2.9 TopicQueries . . . 63
[Major] max_samples_per_remote_writer resource limit not honored in some cases when using an unkeyed Topic and TopicQueries . . . 63
5.2.10 Logging . . . 63
[Major] Unexpected warning during DomainParticipant enablement when locator reachability enabled . . . 63

[Major] Misleading warning message when one non-loopback interface doesn't support multicast . . . 64
[Trivial] Isolated RTIOsapiUtility_strcat error on DomainParticipant creation . . . 64
5.2.11 Dynamic Data . . . 64
[Minor] Unable to set negative values for int8 union discriminators when using Dynamic Data . . . 64
5.2.12 APIs (C or Traditional C++) . . . 65
[Minor] Traditional C++ DDS_KeyedString and DDS_KeyedOctets constructors allowed 0 for a string size . . . 65
5.2.13 APIs (Java) . . . 65
[Critical] Java keyhash calculation error with mutable types in XCDR . . . 65
5.2.14 APIs (Python) . . . 65
[Major] Enhanced logging support in RTI Connext Python API with output_handler_extended . . . 65
5.2.15 APIs (Multiple Languages) . . . 66
[Major] The result of cdr_serialized_sample_max_size was too small if the maximum size was greater than 4 GB . . . 66
[Major] Non-Java register_type failed if the maximum size was greater than 2 GB . . . 66
[Minor] Enum label not printed when there was only one enum member . . . 66
[Minor] Inconsistent QoS error when setting builtin discovery channels . . . 67
5.2.16 XML Configuration . . . 67
[Trivial] Unexpected messages printed in console when there was an error parsing XML application configuration file . . . 67
[Major] Union types loaded from XML did not set ID values correctly when using autoid="hash" . . . 67
[Major] XML namespace parsing error in QoS profiles . . . 67
5.2.17 Entities . . . 67
[Major] Unable to create DistLogger singleton if remote administration was disabled . . . 67
5.2.18 Crashes . . . 68
[Critical] Potential stack overflow in Connext applications when parsing XML files with a custom DTD with deeply nested entity references . . . 68
[Critical] Possible crash when accessing information about a remote endpoint at the same time that the endpoint was being removed . . . 68
[Major] Changes in RTI Connext logging behavior for Windows desktop applications . . . 68
[Critical] Modern C++ Distributed Logger may have crashed upon instance finalization 69
[Critical] Segmentation fault when using Network Capture APIs concurrently with DomainParticipantFactory deletion . . . 69
[Critical] Potential crash while printing the log message after an error . . . 69
[Critical] Segmentation fault when a DomainParticipant failed to be created . . . 69
[Critical] Segmentation fault and precondition errors with Monitoring Library 2.0 when Entities were deleted . . . 69
[Critical] Possible crash when metric value changed when Monitoring Library 2.0 was being disabled . . . 70
5.2.19 Hangs . . . 70
[Critical] Calling ignore_participant during a ParticipantBuiltinTopicDataDataReader on_data_available callback triggered by the deletion of a remote DomainParticipant led to a hang or unexpected error message . . . 70

[Critical] Potential deadlock when calling DDS_DomainParticipantFactory_set_qos and DDS_DomainParticipantFactory_finalize_instance . . . 70
[Critical] Potential deadlock when using Network Capture utility APIs concurrently with DDS calls . . . 71
[Critical] Hang of QNX application using shared memory if one of the shared semaphores was in an unusable state . . . 71
[Major] Calling ignore_participant when using the on_data_available callback to process a ParticipantBuiltinTopicData with partial_configuration led to a hang during DomainParticipant deletion . . . 71
5.2.20 Memory Leaks/Growth . . . 72
[Critical] Spike in memory usage when creating a DataWriter using Durable Writer History to restore samples . . . 72
[Major] RTI DDS Spy showed DDS_DataReader_get_matched_publication_participant_data error before leaking memory . . . 72
[Major] DomainParticipantFactory finalization failed even if all the DomainParticipants were deleted . . . 72
[Minor] Memory leak upon Transport TCP creation failure because of invalid configuration . . . 72
5.2.21 Data Corruption . . . 73
[Critical] Serialization errors in Windows when publishing remote debugging telemetry with Monitoring Library 2.0 . . . 73
5.2.22 OMG Specification Compliance . . . 73
[Major] Incorrect value of an infinite duration in non-Java APIs . . . 73
[Minor] Consistency between max_samples_per_instance and max_samples was not enforced for unkeyed topics . . . 74
5.2.23 Vulnerabilities . . . 74
[Critical] Potential heap buffer read overflow in Connext applications when using a malicious license string . . . 74
[Critical] Potential invalid read memory access in Connext applications when subscribing to PublicationBuiltinTopicData . . . 75
[Critical] Potential invalid read memory access in Connext applications during endpoint discovery . . . 76
5.2.24 Other . . . 76
[Critical] Possible Connext validation error while using a valid license . . . 76
[Critical] Data races on architectures with weak memory models . . . 77
[Major] DomainParticipantFactoryQos could not be retrieved from QosProvider . . . 77
[Major] Incorrect hexadecimal formatting for uint8 and octet . . . 77
[Major] Printing to XML missed quotes for external union members . . . 78
[Major] Printing an IDL or XML enum with a default literal produced incorrect output 78
[Major] Printing an enum default literal printed the ordinal value instead of enumerator name . . . 78
[Major] XML parser rejected external members with ranges excluding zero . . . 79
[Major] Incorrect string representation of union DynamicType with arrays of sequences 80
[Major] Error parsing an XML with an external typedef . . . 81
[Major] Invalid escaping of special characters in default strings for XML and IDL output 81
[Major] DynamicType with default string value incorrectly formatted for XML format 81
[Minor] Multiple attachments of same Condition to a WaitSet . . . 81

[Major] Error when parsing an XML with a struct that has an enum member with a default enumerator . . . 82
[Major] XML string literal interpretation divergence between Core Libraries and Code Generator . . . 82
[Major] XML parser did not parse scientific notation when exponent included an explicit positive sign . . . 82
[Major] Error when parsing an XML with a union that has a member with a default value 83
[Minor] Building a Connext application with Wundef warning category could generate unexpected warnings . . . 83
5.3 What's New in 7.5.0 . . . 83
5.3.1 Python RPC API now fully productized and supported . . . 84
5.3.2 More efficient endpoint discovery for RPC and Request-Reply . . . 85
5.3.3 Improved DataWriter queue sample memory management eliminates required configuration for unbounded types . . . 86
5.3.4 Enhanced out-of-the-box network performance with updated socket buffer sizes . . . 86
5.3.5 Improved scalability for applications using writer-side content filtering . . . 87
5.3.6 Initial discovery performance improvements in large systems with participants containing many endpoints . . . 87
5.3.7 Eliminated requirement to manually clean up thread resources on platforms that support automatic thread-specific storage (TSS) cleanup . . . 87
5.3.8 Improved flow control of fragmented sample repairs . . . 88
5.3.9 RTPS messages containing a malformed inline Qos now trigger a warning, to avoid deserialization errors downstream . . . 88
5.3.10 Capture periodic DomainParticipant announcements in DISCOVERY logging category to enhance debugging of discovery . . . 88
5.3.11 Print incompatible QoS policies in a discovery snapshot for improved discovery debugging . . . 89
5.3.12 Updated default value for socket_monitoring_kind to more performant WINDOWS_IOCP value on Windows systems . . . 89
5.3.13 Support for new metadata fields in content filter expressions . . . 89
5.3.14 InstanceHandle now provides a default constructor . . . 89
5.3.15 Parameter type change to more convenient/appropriate size_t in pluggable transport's to_string APIs . . . 90
5.3.16 Deprecations and Removals . . . 90
"delete" method of DynamicDataTypeSupport deprecated . . . 90
dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size removed (except for Java) . . . 90
5.3.17 Third-Party Software Changes . . . 91
5.4 What's Fixed in 7.5.0 . . . 91
5.4.1 Discovery . . . 91
[Critical] Undefined behavior when serializing using any of the builtin discovery plugins 91
[Critical] Incorrect locators used on newly created DataWriters or DataReaders after a Locator Reachability event . . . 91
[Major] DomainParticipants using SPDP2 and fragmentation of the participant configuration messages failed to complete discovery . . . 92
[Major] DomainParticipants failed to announce that endpoint was not reachable when internal clock was monotonic and external clock was real-time . . . 92

[Minor] DomainParticipants using LBED unregistered remote DataWriters when a remote participant was deleted instead of deleting them . . . 92
5.4.2 Usability . . . 92
[Minor] Unnecessary QoS consistency checks for best effort DataWriters . . . 92
5.4.3 Transports . . . 93
[Critical] Shared Memory communications might not have been recovered if a DataReader crashed or was ungracefully killed . . . 93
5.4.4 Reliability Protocol and Wire Representation . . . 93
[Critical] Writer-side filtered samples not always marked as acknowledged when application acknowledgment was used . . . 93
5.4.5 Performance and Scalability . . . 93
[Critical] High CPU usage and delayed events in a DataReader managing many instances of a deleted DataWriter . . . 93
5.4.6 Debuggability . . . 94
[Major] Incorrect source port for some Real-Time WAN Transport packets recorded with network capture . . . 94
[Trivial] RTI DDS Spy showed unprintable character in help output . . . 94
5.4.7 Content Filters and Query Conditions . . . 94
[Minor] Setting parameters on QueryCondition could have led to slightly inefficient refiltering . . . 94
5.4.8 Request-Reply and Remote Procedure Calls . . . 95
[Minor] Request-reply and RPC readers created without explicitly loading BuiltinQosLib::Pattern.RPC did not set reliability.max_blocking_time to 10 seconds 95
5.4.9 Logging . . . 95
[Trivial] Incorrect informational log messages when adding or removing remote writers in local readers . . . 95
5.4.10 APIs (C or Traditional C++) . . . 96
[Minor] Incorrect return code for DomainParticipant::assert_liveliness API on disabled participant . . . 96
5.4.11 APIs (Java) . . . 96
[Major] Unnecessary memory operations when receiving inherited bounded sequences 96
[Major] For types with strings, sample size calculation in Java may have overestimated serialized size . . . 96
5.4.12 APIs (C# API) . . . 96
[Trivial] .NET bundle information not displayed in Launcher . . . 96
5.4.13 APIs (Python) . . . 97
[Critical] Incompatibility between Python API and other APIs when using int8 type . 97
[Major] Python API didn't support reading TopicQuery samples exclusively . . . 97
[Minor] Python API didn't provide enumeration to set Real-Time WAN Transport in code . . . 97
5.4.14 APIs (Multiple Languages) . . . 97
[Minor] Some properties no longer accepted LENGTH_UNLIMITED string as a valid value . . . 97
[Minor] Request-reply and RPC may have used standard PID instead of selected one . 98
5.4.15 XML Configuration . . . 98
[Minor] Possible failure when loading an XML file containing a long comment . . . 98
5.4.16 Instances . . . 98
[Major] Failure to register instances of mutable flat data types . . . 98

5.4.17 Crashes . . . 99
[Critical] Stack smashing error when serializing strings with RTI_CDR_SIZEOF_LONG_DOUBLE set to 16 in C++11 in release mode using GCC compiler . . . 99
[Critical] Possible crash on MIGInterpreter_parse while receiving data at the same time that entities are removed, on architectures with a weak memory model 99
[Critical] Race condition in RTI Monitoring Library 2.0 caused a crash . . . 99
[Critical] Segmentation fault if Application Kind contained more than 255 characters when calling RTI_DL_DistLogger_getInstance . . . 99
[Critical] Potential crash if creation of Publisher or Subscriber failed while database garbage collection process was running . . . 100
[Critical] Possible crash when calling any write operation on architectures with a weak memory model . . . 100
[Critical] Stack overflow when parsing very long SQL filter expressions . . . 100
[Critical] Possible crash when creating DomainParticipants concurrently . . . 100
[Critical] Crash when writing data on a ContentFilteredTopic in combination with transient durability and delegated reliability . . . 101
[Major] Possible crash after memory allocation failures . . . 101
[Major] Using unbounded or very large types with query conditions in Python API could cause crash . . . 101
[Major] Segmentation fault if application ran out of memory while generating a log message . . . 101
5.4.18 Hangs . . . 101
[Critical] Potential deadlock when using TCP Transport on Windows systems if WINDOWS_WAITFORMULTIPLEOBJECTS was selected as the socket monitoring API . . . 101
[Major] Potential deadlock when using Durable Writer History if there was a failure while creating the connection . . . 102
[Major] Potential deadlock when using DynamicData in Debug libraries if TypePlugin initialization failed . . . 102
[Major] Potential issues using rti.asyncio.run or asyncio.run with dds.DataReader.take_async . . . 102
[Major] RPC failures when serializing the return value or out/inout parameters caused a timeout on client side . . . 102
5.4.19 Memory Leaks/Growth . . . 103
[Critical] Possible unbounded memory growth on DataReader of keyed type after failing to match with a remote DataWriter . . . 103
[Critical] Memory leaks and errors when using DynamicDataReaders or FlatData DataReaders plus DDS-fragmentation and compression or encryption . . . 103
[Critical] Unbounded memory growth when using DynamicDataReaders with multiple data representations and keyed types . . . 103
[Critical] Unbounded memory growth when using RECOVER instance state consistency and instances were removed from DataReader queue . . . 104
[Major] Memory leak when parsing invalid module in XML . . . 104
[Major] Memory leak when using DynamicData with types containing optional members of string aliases or when cloning such types . . . 104
[Minor] DynamicDataTypeSupport objects not reclaimed by garbage collector . . . 105
[Minor] Potential DynamicData sample leaked on DataReader deletion . . . 105

5.4.20 Data Corruption . . . 105
[Critical] Potential corruption in samples with sequences of strings when using Traditional C++ API and std::string . . . 105
[Critical] Rare race condition may have led to undefined behavior . . . 106
[Major] Possible data corruption when using ISerializer<T> to serialize a sample . . . 106
5.4.21 Entities . . . 106
[Major] Error when setting QoS on Publisher or Subscriber without endpoints . . . 106
5.4.22 Interoperability . . . 106
[Critical] XCDR serialization of a mutable union with a discriminator value that does not select a union member was not compliant with OMG specification . . . 106
[Critical] RTPS messages containing vendor-specific submessages could have been dropped . . . 107
[Critical] Multiple RTPS messages received in a single datagram caused issues . . . 107
5.4.23 Vulnerabilities . . . 107
[Critical] Potential stack buffer overflow in Connext applications when parsing an XML type . . . 107
[Critical] Potential stack buffer write overflow in license-managed Core Libraries when setting RTI_LICENSE_FILE environment variable . . . 108
[Critical] Buffer write overflow while parsing malicious license file . . . 109
[Critical] Potential integer overflow in Connext applications on 32-bit systems when parsing XML files with very large number of default attributes or levels of nesting . . . 110
5.4.24 Other . . . 111
[Critical] Data races on architectures with weak memory models . . . 111
[Major] Printing an IDL union with an enum discriminator could have produced an incorrect output . . . 111
[Major] Incorrect CPU assignment when using THREAD_SETTINGS_CPU_RR_ROTATION in receiver_pool . . . 111
[Trivial] Possible data race when creating DomainParticipants . . . 111
[Trivial] Printing a type with an annotation using an enum produced an incorrect output 111
5.5 What's New in 7.4.0 . . . 112
5.5.1 Set up reliable communications for fragmented data more easily and with better performance, by removing asynchronous publishing requirement . . . 112
5.5.2 Modern C++ RPC API now fully productized and supported . . . 113
5.5.3 Discover relevant QoS more easily through categorizations added to API Reference HTML documentation . . . 114
5.5.4 Easier QoS configuration for Request-Reply and RPC with new built-in QoS profile . 115
5.5.5 Write Python applications faster with reactive subscriptions . . . 116
5.5.6 New Ping built-in type helps debug and prototype with the Python API . . . 117
5.5.7 DDS Spy now prints full timestamps (with date and fractions of a second) and epoch timestamps . . . 117
5.5.8 New DynamicData APIs get and set a union's discriminator value . . . 118
5.5.9 Deprecations and Removals . . . 118
Simplify QoS configuration by deprecating unnecessary or rarely used QoS settings and policies . . . 119
dds.sample_assignability.accept_unknown_enum_value and dds.sample_assignability.accept_unknown_union_discriminator properties deprecated . . . 119
dds.participant.use_45d_compatible_alignment_enforcement property deprecated . . 119

EXCLUSIVE_AREA QoS policy removed . . . 119
Deprecated behavior in DynamicData API . . . 120
DDS-prefixed enumerations removed from XSD files . . . 120
5.6 What's Fixed in 7.4.0 . . . 120
5.6.1 Discovery . . . 120
[Critical] DNS Tracking not supported with participants using SPDP2 . . . 120
5.6.2 Serialization and Deserialization . . . 121
[Major] Samples containing sequences using discontiguous buffers failed to be serialized 121
[Major] Calculated maximum size incorrect for unions with nested unions whose members were empty structures . . . 121
5.6.3 Usability . . . 121
[Major] MultiChannel did not work for synchronous DataWriters with large data in best-effort communications . . . 121
5.6.4 Transports . . . 122
[Major] Potential high CPU usage if network stack was deleted . . . 122
[Minor] Wrong configuration of transport_priority QoS on TCP Transport . . . 122
5.6.5 Reliability Protocol and Wire Representation . . . 122
[Critical] Repair sample was not sent if the sample was smaller than either transport's message_size_max . . . 122
[Critical] Late-joiner DataReader may have stopped receiving samples from DataWriters using a finite durability.writer_depth . . . 123
[Major] inactivate_nonprogressing_readers did not always inactivate non-progressing readers . . . 123
[Major] Reliable large data performance issues due to redundant fragment repairs . . . 124
[Major] DataWriters with finite durability.writer_depth may have sent repairs over unicast instead of multicast . . . 124
5.6.6 Bandwidth Sensitivity . . . 124
[Minor] Excessive bandwidth consumption with SPDP2 for peers that were added multiple times . . . 124
5.6.7 Debuggability . . . 124
[Minor] Receiving large sequence numbers caused DataReader to incorrectly report on_sample_lost() with a negative total_count . . . 124
5.6.8 Content Filters and Query Conditions . . . 125
[Critical] Failure filtering samples on a DataReader when using DynamicData and Zero Copy . . . 125
5.6.9 TopicQueries . . . 125
[Major] Response to snapshot TopicQuery may have missed samples for instances that passed the TopicQuery filter . . . 125
5.6.10 Request-Reply and Remote Procedure Calls . . . 126
[Major] Python RPC API didn't support str arguments . . . 126
5.6.11 Logging . . . 127
[Trivial] Error messages printed if reader responded to heartbeat from remote writer that was partially removed . . . 127
[Trivial] Failure getting the Logger instance during instance creation overwrote the verbosity levels of all logging categories to random levels . . . 128
5.6.12 Dynamic Data . . . 128
[Major] Incorrect DynamicData behavior involving empty structures . . . 128
[Major] DynamicData to and from JSON conversion did not include union discriminators 129

[Minor] DynamicData::get_member_info API returned incorrect member_id in some cases . . . 129
5.6.13 APIs (Modern C++ API) . . . 130
[Major] Possible exception thrown due to incorrect DynamicData move constructor or assignment operator . . . 130
[Major] Modern C++ API was not enabled for x64Vx7SR0630llvm8.0.0.2_rtp . . . 130
[Minor] find_member_by_id returned INVALID_INDEX in some cases . . . 130
[Trivial] SampleProcessor not included by rti/rti.hpp header . . . 131
5.6.14 APIs (Java) . . . 131
[Major] Unnecessary memory operations when receiving bounded sequences . . . 131
5.6.15 APIs (Python) . . . 131
[Major] Exception when printing or converting a sequence of octets to string . . . 131
[Major] Incorrect deserialization of nested union with unknown discriminator . . . 132
[Major] x64Linux Connext Python API not able to use libraries from $NDDSHOME/lib 132
[Minor] QosProvider could not be directly created with QosProviderParams . . . 132
5.6.16 APIs (Multiple Languages) . . . 132
[Major] DynamicData Errors when operating on unions with unknown discriminator values . . . 132
[Minor] Creating a DomainParticipant with a negative domain ID was incorrectly allowed 133
5.6.17 XML Configuration . . . 133
[Major] Could not use multiple <types> tags to create user data types . . . 133
[Minor] Duplicate type names were allowed in an XML file for defining user data types 134
[Trivial] DDS-prefixed enumerations caused confusion in XSD files . . . 135
5.6.18 Crashes . . . 135
[Critical] Potential crash when creating new GuardCondition when low on resources . 135
[Critical] Parsing an XML file with a root “types” tag that contained a “name” attribute led to a segmentation fault . . . 136
[Critical] Parsing of malformed RTPS packages may have led to bus error in some platforms . . . 136
[Critical] Potential race conditions in construction or deletion of Requesters and Repliers (C, Traditional C++, C#, and Python) . . . 136
[Critical] Possible crash when creating dds.DynamicData.Topic with a listener . . . 136
[Critical] Possible crash after a failure to create Monitoring Library 2.0 object . . . 136
[Critical] Possible segmentation fault while enabling a DataWriter that enables batching 137
[Critical] Segmentation fault when copying invalid samples and using Zero Copy transfer over shared memory transport . . . 137
[Critical] Possible crash when calling DDS_DataWriter_get_matched_subscriptions() or DDS_DataReader_get_matched_publications() on architectures with a weak memory model . . . 137
[Critical] Possible crash when using QosProvider to load XML file with syntax `file:///` on Windows systems . . . 138
[Critical] Crash on DDS_DomainParticipantFactory_finalize_instance() when using XML-Based Application Creation . . . 138
[Critical] Parsing of malformed RTPS packets may have led to bus error on some platforms . . . 138
[Critical] Crash if element in initial_peers longer than 72 characters added . . . 138
[Major] Potential crash upon serialization, deserialization, or program shutdown when running out of memory . . . 139

[Major] Possible crash when closing a logger device while it was being used 139
[Major] Crash during custom transport load when running out of memory 139
[Major] Precondition failure or segmentation fault when nonBasicTypeName is the name of a module 139
5.6.19 Hangs 140
[Critical] String built-in type unregister_type API blocked associated DomainParticipant 140
[Critical] Potential deadlock when using SHMEM transport on Linux, macOS, and LynxOS systems and killing one application 140
[Critical] Potential deadlock when creating entities in parallel that use Zero Copy transfer over shared memory transport 141
[Major] Running out of memory during DomainParticipant creation caused DomainParticipantFactory finalization to hang 141
5.6.20 Memory Leaks/Growth 141
[Critical] Parsing XML configuration with large tokens caused uncontrolled resource consumption 141
[Critical] Potential unbounded memory growth when using DNS tracking 142
[Critical] Parsing DTD with many nested/recursive entities could cause uncontrolled resource consumption 142
[Major] Memory leak when parsing malformed XML defined types 142
[Minor] Possible memory leak when initializing and finalizing the DomainParticipantFactory concurrently 142
[Minor] Memory leak when parsing XML QoS profile with unknown base_name . . . 143
[Minor] Memory leak during DomainParticipant creation when running out of memory trying to create the event thread 143
5.6.21 Data Corruption 144
[Major] Parsing of malformed DATA_FRAG submessage may have led to unexpected application behavior 144
5.6.22 OMG Specification Compliance 144
[Major] Not possible to specify how to deserialize a sample with a discriminator that does not select a member or an unknown enum 144
5.6.23 Interoperability 145
[Critical] Incorrect handling of RTPS messages with submessages from different participants 145
[Critical] RTPS checksums incorrect for multiple RTPS messages received in a single datagram 145
5.6.24 Vulnerabilities 145
[Critical] Stack buffer write overflow while parsing malicious license file 145
[Critical] Stack buffer write overflow while parsing malicious XML types document . 146
[Critical] Stack buffer write overflow while parsing a malicious XML types document . 147
5.6.25 Other 147
[Critical] Data races on architectures with weak memory model 147
[Major] Missing implementation of RTI_DL_DistLogger_get_version(), RTI_DL_DistLogger_get_api_build_version(), and RTI_DL_DistLogger_get_api_version_string() 148
[Major] Incorrect update of DataWriter's pushed_fragment_count protocol statistic under some conditions 148
[Minor] IDL annotations printed after typdef specifier 148

6 Known Issues **149**
6.1 Known Issues with Discovery . . . 149
6.1.1 SPDP2 . . . 149
Features under future consideration for SPDP2 . . . 149
Participants using SPDP2 and allow_unauthenticated_participants fail to communicate if only one participant fails authentication . . . 150
6.1.2 TypeObject v2 not serialized correctly on big endian platforms . . . 150
6.2 Known Issues with Serialization and Deserialization . . . 150
6.2.1 Some parameters cannot be received multiple times within same SPDP sample . . . 150
6.2.2 Connext not compliant with Extended CDR encoding version 2 for types containing arrays and sequences of non-primitive types . . . 151
6.3 Known Issues with Usability . . . 151
6.3.1 Cannot open USER_QOS_PROFILES.xml in rti_workspace/examples from Visual Studio . . . 151
6.3.2 DataWriter's Listener callback on_application_acknowledgment() not triggered by late-joining DataReaders . . . 152
6.3.3 HighThroughput and AutoTuning built-in QoS Profiles may cause communication failure when writing small samples . . . 152
6.3.4 Memory leak if Foo:initialize() called twice . . . 152
6.3.5 Wrong error code after timeout on write() from Asynchronous Publisher . . . 153
6.3.6 Type Consistency enforcement disabled for structs with more than 10000 members . 153
6.3.7 Escaping special characters in regular/filter expressions not supported in some cases . 153
6.3.8 Non-English Unicode (UTF-8) file names and paths not supported . . . 154
6.4 Known Issues with Code Generation . . . 154
6.4.1 Examples and generated code for Visual Studio 2017 and later may not compile (Error MSB8036) . . . 154
6.5 Known Issues with Instance Lifecycle . . . 154
6.5.1 RECOVER_INSTANCE_STATE_CONSISTENCY setting not fully supported by RTI Infrastructure Services . . . 154
6.5.2 Persistence Service DataReaders ignore serialized key propagated with dispose updates 155
6.5.3 instance_state_consistency_kind QoS cannot be modified before containing entity is enabled . . . 155
6.6 Known Issues with Reliability . . . 155
6.6.1 DataReaders with different reliability kinds under Subscriber with GROUP_PRESENTATION_QOS may cause communication failure . . . 155
6.7 Known Issues with Content Filters and Query Conditions . . . 156
6.7.1 Writer-side filtering may cause missed deadline . . . 156
6.7.2 filter_sample_* statistics in DDS_DataWriterProtocolStatus not updated correctly . . 156
6.8 Known Issues with TopicQueries . . . 156
6.8.1 TopicQueries not supported with DataWriters configured to use batching or Durable Writer History . . . 156
6.9 Known Issues with C# API . . . 156
6.9.1 Connext API . . . 157
6.9.2 IDL-to-C# Code Generation . . . 157
6.9.3 API Reference . . . 157
6.10 Known Issues with Transports . . . 157
6.10.1 AppAck messages cannot be greater than underlying transport message size . . . 157
6.10.2 DataReader cannot persist AppAck messages greater than 32767 bytes . . . 158

6.10.3 Discovery with Connext Micro fails when shared memory transport enabled 158
6.10.4 Communication may not be reestablished in some IP mobility scenarios 158
6.10.5 Corrupted samples may be forwarded through Routing Service when using Zero-Copy transfer over shared memory 159
Use automatic application acknowledgment 159
Ensure that the number of available samples accounts for Routing Service processing time 159
6.10.6 Network Capture does not support frames larger than 65535 bytes 159
6.10.7 Shared memory transport in QNX 7.0 and earlier can result in priority inversion . . . 160
6.10.8 Ungracefully terminated QNX processes using SHMEM transport prevents startup of new processes due to unclosed POSIX semaphores (QNX 7.0 and earlier) 160
6.11 Known Issues with FlatData 160
6.11.1 FlatData language bindings do not support automatic initialization of arrays of primitive values to non-zero default values 160
6.11.2 Flat Data: plain_cast on types with 64-bit integers may cause undefined behavior . . 161
6.12 Known Issues with Coherent Sets 161
6.12.1 Some coherent sets may be lost or reported as incomplete with batching configurations 161
6.12.2 Copy of SampleInfo::coherent_set_info field is not supported 161
6.12.3 Other known issues with coherent sets 161
6.13 Known Issues with Dynamic Data 162
6.13.1 Conversion of data by member-access primitives limited when converting to types that are not supported on all platforms 162
6.13.2 Types that contain bit fields not supported 162
6.13.3 Long double not supported for DynamicData in Java API 162
6.14 Known Issues with RTI Monitoring Library 162
6.14.1 Problems with NDDS_Transport_Support_set_builtin_transport_property() if Participant Sends Monitoring Data 163
6.14.2 Participant's CPU and memory statistics are per application 163
6.14.3 ResourceLimit channel_seq_max_length must not be changed 163
6.15 Known Issues with RTI Monitoring Library 2.0 163
6.15.1 Slow DDS Entity creation in large applications 163
6.16 Other Known Issues 164
6.16.1 Possible Valgrind still-reachable leaks when loading dynamic libraries 164
6.16.2 64-bit discriminator values greater than (2^31-1) or smaller than (-2^31) not supported 164
6.16.3 Creating multiple DataReaders for the same Topic under the same Subscriber configured with Group Ordered Access is not supported 164
6.16.4 With DISALLOW_TYPE_COERCION and Types containing unbounded members, other vendor DataWriters/DataReaders will not match Connext DataWriters/DataReaders 165
6.16.5 find_datawriters/readers APIs when used with XML-Based Application Creation is not supported on systems that use older GCC versions 165

7 Copyrights and Notices **166**

Chapter 1

System Requirements

Connext requires a multi-threaded operating system. This section describes the supported host and target systems.

In this context, a host is the computer on which you will be developing a *Connext* application. A target is the computer on which the completed application will run. A host installation provides the *RTI Code Generator* tool (*rtiddsgen*), examples and documentation, as well as the header files required to build a *Connext* application for any architecture. You will also need a target installation, which provides the libraries required to build a *Connext* application for that particular target architecture.

Supported platforms, for all products in the *Connext* suite, are listed in *Supported Platforms*.

Future releases may not support all of the platforms supported in this release, or may support different versions of platforms supported in this release.

1.1 Supported Platforms

A *platform* refers to the combination of your target machine's OS version, CPU, and toolchain (compiler or Visual Studio). For example, if you have a 64-bit Windows machine with Visual Studio® 2017, the architecture name is `x64Win64VS2017`. For a 64-bit Linux machine with gcc version 8.5.0, the architecture name is `x64Linux4gcc8.5.0`.

Table 1.1 *RTI Architecture Names* lists the operating systems and their architecture names supported in this release. You will need to know your architecture name when downloading/installing various *Connext* libraries. (For platform changes in this release, see Platform and Build Changes in the *Connext What's New*.)

Note: For full details on these platforms, including what languages, features, and products they support, see the RTI Connext Core Libraries Platform Notes.

Table 1.1: RTI Architecture Names

	Operating System	OS Version	CPU	Compiler/Toolchain	RTI Architecture
Standard Target Libraries	Android	Android 12, 14	ARM64	clang 12.0.8 (ndk r23b)	arm64Android12clang12.0.8ndkr23b
		AOSP 14	ARM64	clang 17.0.2	arm64AndroidAOSP14clang17.0.2
	Linux® (Intel)	Red Hat® Enterprise Linux 8, 9, 10; Ubuntu 22.04 LTS, 24.04 LTS	x64	gcc 8.5.0	x64Linux4gcc8.5.0
	Linux (Arm®)	Ubuntu 22.04 LTS, 24.04 LTS	ARMv8	gcc 8.5.0	armv8Linux4gcc8.5.0
	macOS®	macOS 14, 15, 26 (Darwin 23, 24, 25)	ARM64	clang 16.0, 17.0	arm64Darwin23clang16.0
	QNX	QNX Neutrino 8.0, 8.0.2, 8.0.3	x64	qcc_cxx 12.2.0 (LLVM C++ library)	x64QNX8.0qcc_cxx12.2.0
			ARMv8	qcc_cxx 12.2.0 (LLVM C++ library)	armv8QNX8.0qcc_cxx12.2.0
	Windows® (Intel®)	Windows 11; Windows Server 2022	x64	VS2017, VS2019, VS2022	x64Win64VS2017
Docker Images	Linux (Intel)	Ubuntu 24.04 LTS	amd64	gcc 8.5.0	x64Linux4gcc8.5.0
	Linux (ARM)	Ubuntu 24.04 LTS	arm64	gcc 8.5.0	armv8Linux4gcc8.5.0
Debian Packages	Linux (Intel)	Debian 11 (bullseye), Debian 12 (bookworm)	amd64	gcc 8.5.0	x64Linux4gcc8.5.0
	Linux (ARM)	Debian 11 (bullseye), Debian 12 (bookworm)	arm64	gcc 8.5.0	armv8Linux4gcc8.5.0

Custom-supported target platforms are available on demand. Contact sales@rti.com.

1.2 Disk and Memory Usage

Disk usage for a typical installation is approximately 3-5 GB depending on the number of targets and packages installed.

The memory requirements for your target system depend on the complexity of your application and hardware architecture. For more information, see the "Memory Consumption" section for the latest LTS release in the Migration Guide.

Chapter 2

Compatibility

Below is basic compatibility information for this release.

Note: For backward-compatibility information between this and previous releases, see the *Migration Guide* on the RTI Community Portal.

2.1 Wire Protocol Compatibility

Connext communicates over the wire using the formal Real-Time Publish-Subscribe (RTPS) protocol. RTPS has been developed from the ground up with performance, interoperability and extensibility in mind. The RTPS protocol is an international standard managed by the OMG. The RTPS protocol has built-in extensibility mechanisms that enable new revisions to introduce new message types, extend the existing messages, or extend the Quality of Service settings in the product—without breaking interoperability.

RTPS 1.0 was introduced in 2001. The currently supported version is OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5, although some features are not supported. Unsupported features currently are FilteredCountFlag in GAP Submessage, HeartbeatFrag Submessage, and ALIVE_FILTERED instance state. RTI plans to maintain interoperability between middleware versions based on RTPS 2.1. For more details, see Table 2.1 *RTPS Versions*.

Table 2.1 *RTPS Versions* shows RTPS versions supported for each *Connext* release. In general, RTPS 2.1 and higher versions are interoperable, unless noted otherwise. RTPS 2.0 and RTPS 1.2 are incompatible with current (4.2e and later) versions of *Connext*.

Although RTPS 2.1 and higher versions are generally interoperable, there may be specific wire protocol interoperability issues between *Connext* releases. These issues are documented in the "Wire Protocol" section for your release, in the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation). Wire protocol issues between 5.3.1 and previous releases are documented in the *Core Libraries Release Notes* for release 5.3.1.

Table 2.1: RTPS Versions

Connext Release	RTPS Standard Version[Page 5, 1]	RTPS Protocol Version[2]
Connext 7.1.0 and above	2.5 (partial support)	2.5
Connext 6 and 7.0.0	2.3 (partial support)	2.3
Connext 5.2 and 5.3	2.2	2.1
Connext 4.5f - 5.1	2.1	2.1
Data Distribution Service 4.2e - 4.5e	2.1	2.1
Data Distribution Service 4.2c	2.0	2.0
Data Distribution Service 4.2b and lower	1.2	1.2

2.2 Code and Configuration Compatibility

The *Connext* core uses an API that is an extension of the OMG Data Distribution Service (DDS) standard API, version 1.4. RTI strives to maintain API compatibility between versions, but will conform to changes in the OMG DDS standard.

The *Connext* core primarily consists of a library and a set of header files. In most cases, upgrading simply requires you to recompile your source using the new header files and link the new libraries. In some cases, minor modifications to your application code might be required; any such changes are noted in the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation). The *Migration Guide* also indicates whether and how to regenerate code.

2.3 Extensible Types Compatibility

This release of *Connext* includes partial support for the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.4[3] (DDS-XTypes) from the Object Management Group (OMG). This support allows systems to define data types in a more flexible way, and to evolve data types over time without giving up portability, interoperability, or the expressiveness of the DDS type system.

For information related to compatibility issues associated with the Extensible Types support, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation). See also the *RTI Connext Core Libraries Extensible Types Guide* for a full list of the supported and unsupported extensible types features.

[1] Version number of the RTPS standards document, OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5

[2] RTPS wire protocol version number that *Connext* announces in messages it puts on the wire

[3] At the time of the 7.7.0 release, version 1.4 of the OMG 'Extensible and Dynamic Topic Types for DDS' specification has been approved but not yet been made public. It should be available soon after the release.

Chapter 3

What's New in 7.7.0

This section describes what's new in the Core Libraries compared to release 7.6.0 (see *Previous Releases*).

RTI® Connext® 7.7.0 is the first production release in the 7.7 long-term support (LTS) series. It is built upon and combines all of the features in releases 7.4.0, 7.5.0, and 7.6.0. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation. Or start with the RTI Connext What's New for a launchpad to all products.

Note: For backward compatibility information between 7.7.0 and previous releases, see the Migration Guide on the RTI Community Portal.

3.1 Discovery

3.1.1 Reduce network traffic and system startup time by sending full type information only when needed

Connext now provides support for the **OMG Extensible Types TypeLookup Service** and **TypeObject v2** wire representation, as defined by the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.4. This feature addresses discovery scalability problems common in large-scale and evolving systems by fundamentally changing how data types are shared across a network.

Previously, *Connext* propagated full type definitions (TypeObject v1) from every endpoint to every application during the discovery process. In systems with hundreds of nodes or complex types, this resulted in massive redundant wire traffic and slow discovery times.

The new TypeLookup Service introduces a more intelligent, request-reply mechanism for type propagation, providing on-demand type discovery. Instead of blindly sending full type definitions, endpoints now send a unique **TypeIdentifier (hash)** during initial discovery. As a result, type information is sent only a single time when needed.

- **Hash-Based Identification:** Participants use these hashes to check for local compatibility without needing the full type description.
- **On-Demand Retrieval:** If a participant discovers an unknown hash, it uses a dedicated, reliable builtin channel to request the specific TypeObject v2 definition only once.
- **Efficient Propagation:** Once a type is retrieved, it is cached locally, ensuring that the same definition is never sent more than once to any single participant.

Key benefits:

- **Reduced Network Traffic:** Greatly minimizes the amount of data sent during discovery, particularly in large systems where the same type is reused across many endpoints.
- **Faster System Startup:** Improves discovery completion time, allowing applications to reach a "ready" state much more quickly.
- **Richer Type Metadata:** TypeObject v2 is more expressive than its predecessor, allowing the propagation of previously unsupported information such as member ranges (@min, @max) and default values (@default).
- **Vendor Interoperability:** Enhances out-of-the-box compatibility with other DDS implementations that rely on the standard TypeLookup Service to resolve type dependencies.

The TypeLookup Service and TypeObject v2 are **enabled by default** in this release. To maintain compatibility with older *Connext* versions (which use TypeObject v1), newer applications can be configured to send both v1 and v2 representations simultaneously or just TypeObject v1. Additionally, customers who previously disabled type propagation entirely (by setting `type_object_max_serialized_size` to 0) can continue to do so with no performance regressions.

See Type Lookup Service in the *RTI Connext Core Libraries User's Manual* for more information.

3.1.2 type_object_max_serialized_length defaults to LENGTH_AUTO, in support of new TypeObject v2 default

Both `type_object_max_serialized_length` and `serialized_type_object_dynamic_allocation_threshold` now have default values of LENGTH_AUTO. LENGTH_AUTO disables the propagation of TypeObject v1, in favor of the new default, TypeObject2. (LENGTH_AUTO effectively translates to 0 internally for both `type_object_max_serialized_length` and `serialized_type_object_dynamic_allocation_threshold`.)

See the RTI Connext Core Libraries Extensible Types Guide for more information.

3.1.3 LBED Plugin now supports setting DataRepresentationQosPolicy

The *RTI Limited Bandwidth Endpoint Discovery (LBED) Plugin* now supports the **DataRepresentation** QoS policy when asserting remote endpoints. Previously, if the **DataRepresentation** QoS policy was set in the LBED XML configuration, its value was ignored and the default value was assumed for all remote endpoints. This meant that if the value for local endpoints was set to something that was not compatible with the default value, discovery would fail due to incompatible QoS.

For a full list of the supported QoS Policies, see 4.3 Supported QoS in the *RTI LBED Plugin User's Manual*.

3.2 Performance and Scalability

3.2.1 Significant memory footprint reduction for Connext applications

Release 7.7.0 delivers substantial memory savings for *Connext* applications, reducing usage for large-scale systems between 15% to 40% based on certain QoS settings and application size.

These optimizations are organized into two areas:

- **General improvements** that benefit all *Connext* applications, regardless of which features are in use.
- **Feature-specific improvements** that lower memory usage based on the application's configuration:
  - Applications leveraging the *Security Plugins* will see a reduction in overall memory consumption.
  - When a reliable *DataWriter* is matched exclusively with reliable *DataReaders*, setting `writer_qos.reliable.exact_reliability_kind_match` to true (see here) avoids allocating unnecessary resources on the *DataWriter* side.
  - *DataReaders* configured to filter redundant samples (the default) now consume less memory.
  - Memory usage on the *DataReader* side is automatically reduced when the Request/Reply pattern or *Application-Level Acknowledgments* are not in use.
  - Beyond the lazy allocation introduced by the new QoS field `reader_qos.reader_resource_limits.lazily_allocate_batch_resources` (see here), overall memory consumption for batching on the *DataReader* side has been optimized.

These changes are especially valuable for large distributed systems where many participants and endpoints drive rapid memory growth, for edge and embedded platforms (such as robotics, medical devices, and autonomous systems) that operate under tight memory constraints, and for cloud or virtualized environments where a smaller footprint directly translates to cost savings and higher workload density.

See also RTI Connext Performance Benchmarks.

3.2.2 Reduced default metrics set in Monitoring Library 2.0, minimizing bandwidth utilization

When a *Collector Service* is active, *Monitoring Library 2.0* now reduces the default set of published metrics to minimize bandwidth utilization.

The default metrics focus on those required to drive alerts in the standard Grafana Observability dashboards. Additional metrics can be enabled on demand from the Grafana UI by sending a command to the *Collector Service*.

To see which metrics are enabled by default, refer to Setting Initial Metrics and Log Configuration in the *RTI Connext Observability Framework User's Manual*.

To restore the previous behavior and publish all available metrics, update the *Monitoring Library 2.0* configuration as follows:

```
<qos_library name="MyQosLibrary">
    <qos_profile name="MyApplicationProfile" is_default_participant_factory_
↪profile="true">
        <participant_factory_qos>
            <monitoring>
                <!-- Enable monitoring -->
                <enable>true</enable>
                <!-- Enable all metrics -->
                <telemetry_data>
                    <metrics>
                        <element>
                            <resource_selection>//*</resource_selection>
                            <enabled_metrics_selection>
                                <element>*</element>
                            </enabled_metrics_selection>
                        </element>
                    </metrics>
                </telemetry_data>
            </monitoring>
        </participant_factory_qos>
    </qos_profile>
</qos_library>
```

See Monitoring Library 2.0 in the *RTI Observability Framework User's Manual* for more information.

3.2.3 Improved discovery and reduced overhead with Collector Service

Default discovery and liveliness settings for *Collector Service* and *Monitoring Library 2.0* participants have been updated to improve responsiveness.

Because observability discovery is asymmetric (applications discover *Collector Service*), late-joining *Collector Services* could previously take up to 30 seconds to be discovered. This is now improved.

The new defaults provide faster discovery and quicker detection of *Collector Service* loss:

- `participant_liveliness_lease_duration` = 35s

- `participant_liveliness_assert_period` = 10s
- `max_liveliness_loss_detection_period` = 10s

In addition, bandwidth has been reduced by disabling unnecessary system properties.

Several system properties that increase DATA(P) size are now disabled by default, reducing bandwidth and memory usage. This disabled properties are as follows:

- `dds.sys_info.creation_timestamp`
- `dds.sys_info.executable_filepath`
- `dds.sys_info.execution_timestamp`
- `dds.sys_info.target`
- `dds.sys_info.username`

Finally, the initial discovery announcements on LAN (`<initial_participant_announcements>`) have decreased from 5 to 3. Lowering the number of initial participant announcements reduces discovery traffic to the *Monitoring Library 2.0* initial peers.

All these changes are part of the `Generic.Monitoring2` builtin profile. See Configuring QoS for Entities in the *RTI Observability Framework User's Manual* for more information.

3.2.4 Improved write execution time when dds.data_writer.history.purge_samples_after_acknowledgment set to 1

This release improves the *DataWriter* `write()` execution time and CPU usage in the following scenario:

- The *DataWriter* enables durable writer history.
- The *DataWriter* is configured with Required Subscriptions.
- The *DataWriter* is configured with dds.data_writer.history.purge_samples_after_acknowledgment set to 1.

The performance improvement is in the writing of live data and is more noticeable when there are many historical samples in the *DataWriter* queue that are being delivered and acknowledged while the *DataWriter* is writing live data.

3.2.5 Durable Writer History performance improvements when application-level acknowledgments enabled

This release may improve the performance of Durable Writer History when application-level acknowledgments are enabled in the *DataWriter* (by setting `reliability.acknowledgment_kind` to `APPLICATION_AUTO_ACKNOWLEDGMENT_MODE` or `APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE`) and when the *DataWriter* does not set the `on_application_acknowledgment` callback in the `DataWriterListener`.

See Notification of Delivery with Application Acknowledgment for more information.

3.2.6 Priority inheritance used when creating semaphores in Linux

Starting with this release, mutexes are created with priority inheritance on Linux platforms that support it. Now, if `ThreadSettingsKind::REALTIME_PRIORITY` is used, there is no risk of priority inversion where a high-priority thread is being indirectly blocked by a lower-spriority thread.

See Using REALTIME_PRIORITY in the *RTI Connext Core Libraries Platform Notes*.

3.2.7 Improved responsiveness to remote reader configuration updates

Updates to remote *DataReader* configurations (for example, ContentFilteredTopic changes) are applied more promptly now, making `DataWriter::get_matched_subscription_data` less likely to block while waiting for those changes to take effect.

3.2.8 Performance improvement loading XML QoS profiles

This release introduces a performance improvement for loading user-defined and builtin XML QoS profiles, particularly in environments with file-system contention or slow I/O. Previously, the process of loading profiles could trigger repeated file-system lookups for the NDDS_DISCOVERY_PEERS file—even when the file did not exist—resulting in unnecessary overhead. With this change, the existence of the NDDS_DISCOVERY_PEERS file is checked only once and the result is cached, significantly reducing redundant file-system operations during profile loading. This enhancement may lead to noticeable performance gains during operations such as setting the *DomainParticipantFactory* QoS or creating a *DomainParticipant*.

3.2.9 Improved performance of Dynamic Data-to-text format functions

The performance of the `DDS_DynamicDataFormatter_to_***()` functions used to convert from a Dynamic Data object into a text-based format has been greatly improved.

3.2.10 DomainParticipants rejoin the system faster

Discovery in asymmetric liveliness loss scenarios has been sped up by sending late-joiner heartbeats immediately in response to late joining (or rejoining) participants. Previously, the response was delayed by the **late_joiner_heartbeat_period** (3s by default), slowing down the return of communication.

3.2.11 Disable reliable writer communication with best effort reader to improve performance and memory usage

Reliable *DataWriters* may send data to both Best Effort and Reliable *DataReaders*. The Reliability QoS property exact_reliability_kind_match has been added to avoid Reliable *DataWriters* matching with Best Effort *DataReaders*. Set this QoS property to true in the *DataWriter* to save memory and CPU cycles if communication with Best Effort *DataReaders* is not required.

See the reliability properties to know when a *DataReader* and a *DataWriter* will match when the QoS property exact_reliability_kind_match is set to true or false.

3.2.12 Improve performance of durable DataWriters when sending data to late-joining DataReaders, using new property

A new property has been added, `dds.data_writer.history.odbc_plugin.read_batch`, which defines how many DDS samples will be pre-fetched from the database to satisfy requests from late-joiners by a *DataWriter* that is configured with durable writer history.

Lowering this value may help reduce temporary initial memory allocations when connecting to a late-joining *DataReader* when there are a large number of samples in the database.

When a *DataReader* requests DDS samples from a *DataWriter* by sending a NACK message, the *DataWriter* may retrieve additional DDS samples from the database to minimize disk access.

See the Property Reference Guide for information about the property. See also How To Configure Durable Writer History in the *RTI Connext Core Libraries User's Manual*.

3.3 APIs

3.3.1 New resource limit gives more control over memory allocation

Previous releases already had the following resource limit QoS settings:

- DataWriterResourceLimitsQos::max_remote_readers
- DataReaderResourceLimitsQos::initial_remote_writers
- DataReaderResourceLimitsQos::max_remote_writers

This release completes the configuration by adding DataWriterResourceLimitsQos::initial_remote_readers. Now, you have more control over how much memory to pre-allocate vs. dynamically allocate for the remote readers matched with a given *DataWriter*.

3.3.2 New APIs allow your application code to check for type assignability issues

Connext checks by default whether two topic-types are structurally compatible (assignable) in order to decide if a *DataWriter* and *DataReader* with different types can communicate. See Verifying Type Consistency: Type Assignability.

This release exposes this functionality in the API so you can now check whether two types are assignable in your application code. For example, you could write custom monitoring utilities that programmatically compare publication and subscription types to detect type compatibility issues.

The newly added APIs are as follows:

C

```
DDS_Boolean DDS_TypeCode_is_assignable_from(
        const DDS_TypeCode *self,
        const DDS_TypeCode *from,
        DDS_ExceptionCode_t *ex
)
```

Traditional C++

```
DDS_Boolean DDS_TypeCode::is_assignable_from(
        const DDS_TypeCode *from,
        DDS_ExceptionCode_t& ex
) const
```

Modern C++

```
bool dds::core::xtypes::DynamicType::is_assignable_from(
        const DynamicType& from
) const
```

C#

```
public bool DynamicType.IsAssignableFrom(DynamicType from)
```

Python

```
DynamicType.is_assignable_from(from_type: DynamicType) -> bool
```

Note: the Java API already provided `boolean TypeCode.assignable(TypeCode other)`.

3.3.3 Option to lazily allocate batching resources in DataReaders, reducing unnecessary memory usage

Previously, *DataReaders* allocated resources for batch management up front. This resulted in unnecessary memory usage if the *DataReader* never matched with a *DataWriter* that had batching enabled.

To address this, *DataReaders* now default to lazy allocation, meaning memory is allocated from the heap only upon receiving the first batch of samples. While this significantly reduces memory consumption for non-batched use cases, it may introduce slight latency when the first batch is received. If you require deterministic performance and prefer to allocate all resources up front, you can set `DataReaderResourceLimitsQos::lazily_allocate_batch_resources` to FALSE.

See DATA_READER_RESOURCE_LIMITS QosPolicy in the *RTI Connext Core Libraries User's Manual*.

3.3.4 Python API supports versions 3.10 to 3.14, no longer supports versions 3.8 and 3.9

This release adds support for the Connext Python API on Python 3.13 and 3.14. The Python API now runs on versions 3.10 to 3.14. Python 3.8 and 3.9 are no longer supported.

3.4 Usability

3.4.1 Monotonic clock as new default enhances robustness against system clock changes

To enhance robustness against system clock changes, the internal clock now defaults to a monotonic clock on all platforms that support it, rather than to a realtime clock. It defaults to realtime if the platform does not support a monotonic clock. This change helps prevent issues related to adjustments in the system time. See Configuring the Clock per DomainParticipant in the *Core Libraries User's Manual* for more information.

3.4.2 Increased maximum length of command-line parameters for rtiddsspy

The maximum length of the command-line parameters for *rtiddsspy* was limited to 512 bytes. Now the limit is 1024 bytes.

3.4.3 Built-in QoS Snippets added to easily enable or disable TypeObject v1 and TypeObject v2

The following Built-in QoS Snippets were added, to allow you to easily enable or disable TypeObject v1 and TypeObject v2 via XML:

- BuiltinQosLib::Feature.TypeSupport.V1.Enable: Enables TypeObject v1 propagation.
- BuiltinQosLib::Feature.TypeSupport.V1.Disable: Disables TypeObject v1 propagation.
- BuiltinQosLib::Feature.TypeSupport.V2.Enable: Enables TypeObject v2 propagation.
- BuiltinQosLib::Feature.TypeSupport.V2.Disable: Disables TypeObject v2 propagation.

See *Reduce network traffic and system startup time by sending full type information only when needed* for more context. See QoS Profile Inheritance and Composition for information on Snippets.

3.4.4 Flexible control over Monitoring Library 2.0 enablement using new environment variable

A new environment variable, `RTI_MONITORING2_ENABLE`, can now be set to `1`, `true`, or `on` (case insensitive) to enable Monitoring Library 2.0. The variable can be set to `0`, `false`, or `off` to disable it. The `RTI_MONITORING2_ENABLE` setting takes precedence over the XML configuration.

See Enabling Monitoring Library 2.0 in the *RTI Connext Observability Framework User's Manual* for more information.

3.5 Debuggability

3.5.1 Monitoring Library 2.0 now enabled by default under specific conditions, making it easier to use remote debugging

Monitoring Library 2.0 is now enabled by default for *Connext* applications if either of the following conditions is met:

- Your application links to dynamic *Connext* libraries and *Monitoring Library 2.0* is available in your library path.
- Your application links to static *Connext* libraries (including *Monitoring Library 2.0*) and you invoke `RTI_Monitoring_initialize`.

This improvement enables you to use the *Observability Framework* and *Admin Console* Remote Debugging feature at any time without modifying your deployed applications. All *Connext* Infrastructure Services (except *Collector Service*) also enable the library by default when available in the library path. The library remains disabled by default in *RTI Admin Console*, *RTI Monitor*, *RTI DDS Spy*, and *RTI DDS Ping*.

See Enabling Monitoring Library 2.0 in the *Observability Framework* documentation for more information.

3.5.2 Monitoring Library 2.0 security enforced when using an RTI security plugin, elevating security standards

To prevent *Monitoring Library 2.0* from inadvertently publishing telemetry data and metrics from a secured application over an unsecured channel, *Connext* now enforces that *Monitoring Library 2.0* is also secured whenever any *DomainParticipant* in the application uses a security plugin (either the *RTI Security Plugins* or the *Lightweight Security Plugins*).

If any *DomainParticipant* in your application has security enabled and *Monitoring Library 2.0* is not secured, the enablement of *Monitoring Library 2.0* will fail with the following error:

```
[ERROR] FAILED TO ENABLE | You are attempting to enable the RTI Monitoring␣
↪Library 2.0 without security, but your system already contains a␣
↪DomainParticipant with security enabled. Please, enable security for the␣
↪RTI Monitoring Library 2.0 (refer to the RTI Connext Observability␣
↪Framework for further information).
```

Conversely, if *Monitoring Library 2.0* is already enabled without security, enabling a *DomainParticipant* that uses a security plugin will fail with the following error:

```
[ERROR] FAILED TO ENABLE | You are attempting to enable a secure␣
↪DomainParticipant, but the RTI Monitoring Library 2.0 is not secured.␣
↪Please, disable RTI Monitoring Library 2.0 (using factoryQos.monitoring.
↪enable) or enable security for it (refer to the RTI Connext Observability␣
↪Framework for further information).
```

To resolve either error, enable security for *Monitoring Library 2.0* by configuring the appropriate security artifacts in the monitoring *DomainParticipant* QoS profile, or disable *Monitoring Library 2.0* using `participant_factory_qos.monitoring.enable`.

For details on how to secure *Monitoring Library 2.0*, see the Security chapter in the RTI Connext Observability Framework User's Manual.

3.5.3 Improved diagnostics through extended SQLite error codes in durable writer and reader state databases

Errors detected in SQLite database files used for durable writer history and durable reader state now include SQLite extended error codes. This additional information improves diagnostics and helps identify the underlying cause of database-related issues.

3.5.4 Debug symbols for release libraries now provided for Linux platforms

On Linux platforms, debug symbols for the release libraries are now also provided as an alternative to the debug libraries, allowing source-level debugging without the performance overhead of the debug libraries. See Debug Symbols for Release Dynamic Libraries in the *RTI Connext Core Libraries Platform Notes* for more information.

3.6 OMG Specification Compliance

3.6.1 Better control communication in a system with changing types and evolving data model requirements, using @must_understand annotation

Connext now lets you mark members of a struct as "Must-Understand" to make sure that they will be present in both types of two matched endpoints.

In most cases, struct types that are extended with additional members will remain "assignable" to their base versions, which allows *DataWriters* and *DataReaders* that use different versions of a type to still match and exchange data.

During the evaluation of a potential match between a *DataWriter* and a *DataReader*, any member marked as Must-Understand that doesn't appear in the other side's type will cause the two endpoints to have "unassignable" types, and the match to fail.

When using @mutable types, Must-Understand members that are also marked Optional will not affect matching, but any sample containing them will be dropped by any *DataReader* whose type doesn't declare them.

See Must-Understand Members in the *RTI Connext Core Libraries Extensible Types Guide* for more information.

3.6.2 Augment your data models by explicitly describing units of measurement associated with types

You can now quantify your data with the `@unit` annotation, to explicitly describe units of measurement associated with your types. The `@unit` annotation can be used on aliases of any type, and on any member of a struct or union. The annotation accepts an arbitrary string value, without enforcing any specific syntax.

The annotated value can be retrieved at runtime by inspecting the generated type descriptors, and it will be propagated through discovery for access from remote nodes.

See Units in the *RTI Connext Core Libraries Extensible Types Guide* for more information.

3.7 DDS Spy

3.7.1 Improved transport configuration in DDS Spy with alias-based option

RTI DDS Spy now supports configuring built-in transports using human-readable aliases (e.g., `shmem|udpv4`) instead of decimal bitmasks. This enhancement makes it easier to enable and combine *UDPv4*, *UDPv6*, and shared memory transports when using the `-transport` option.

See Usage in the *RTI DDS Spy User's Manual* for more information.

3.7.2 Support for Real-Time WAN Transport in DDS Spy

This release adds support to *DDS Spy* for the *RTI Real-Time WAN Transport*. When using `udpv4_wan` with the `-transport` option, you must always provide a *Real-Time WAN Transport* peer using the `-peer` option.

Example:

```
rtiddsspy -transport udpv4_wan -peer udpv4_wan://10.10.10.1:1234
```

See Usage in the *RTI DDS Spy User's Manual* for more information.

3.7.3 Support for participant discovery events in DDS Spy

This release introduces support for participant discovery events in *DDS Spy (rtiddsspy)*. The utility now reports when participants are discovered, updated, or deleted.

Example output:

```
RTI Connext DDS Spy built with DDS version: 7.7.0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
rtiddsspy is listening for data, press CTRL+C to stop it.
16:48:51 New participant    from 10.30.2.67     : guid="0x010167FB,0xE44F34EB,
↪0xB6EF158B:0x000001C1" name="Shapes Role:RTI Shapes Demo" partition="PartA,
↪PartB" processId="12234"
16:48:51 Updated participant from 10.30.2.67     : guid="0x010167FB,
↪0xE44F34EB,0xB6EF158B:0x000001C1" name="Shapes Role:RTI Shapes Demo"␣
↪partition="PartA,PartB" processId="12234"
16:48:54 New writer         from 10.30.2.67     : topic="Square" type=
↪"ShapeType" guid="0x010167FB,0xE44F34EB,0xB6EF158B:0x80000002"␣
↪publisherObjectId="0x00000108" name="RTI Shapes Demo" DP partition="PartA,
↪PartB" P  partition="Part1"
17:46:46 Deleted writer     from 10.30.2.67     : topic="Square" type=
↪"ShapeType" guid="0x010167FB,0xE44F34EB,0xB6EF158B:0x80000002"␣
↪publisherObjectId="0x00000108" name="RTI Shapes Demo" DP partition="PartA,
↪PartB" P  partition="Part1"
17:46:48 Deleted participant from 10.30.2.67     : guid="0x010167FB,
```

```
↪0xE44F34EB,0xB6EF158B:0x000001C1" name="Shapes Role:RTI Shapes Demo"␣
↪partition="PartA,PartB" processId="1234"
```

See Understanding the Output in the *RTI DDS Spy User's Manual* for more information.

3.7.4 DDS Spy option for printing Publisher and Subscriber Object IDs for easier diagnostics

This release introduces an improvement that, when the existing command-line option `-showHandle` is used in *rtiddsspy*, prints the Object ID of the *Publisher* and *Subscriber* associated with a discovered entity (*DataWriter* or *DataReader*), in addition to the entity GUID. This capability enhances traceability, making it easier to analyze and troubleshoot your applications.

For example:

```
16:39:01 New writer        from 192.168.1.213  : topic="Square" type=
↪"ShapeType" guid="0x0101850C,0x604B1633,0x2129819D:0x80000002"␣
↪publisherObjectId="0x00000108" name="RTI Shapes Demo"
```

In this example, the *Publisher* GUID shares the prefix with the entity GUID and differs only in the Object ID: `0x0101850C,0x604B1633,0x2129819D:0x00000108`.

See Usage in the *RTI DDS Spy User's Manual* for more information.

3.7.5 New DDS Spy option for QoS inspection to help identify incompatibilities that prevent communication

This release introduces a new command-line option, `-showEntityQos`, for *DDS Spy (rtiddsspy)* that allows you to inspect the QoS policy values used for matching in discovered entities. Being able to inspect QoS values used helps identify incompatibilities that prevent communication.

The displayed QoS policies include:

- Reliability
- Durability
- Deadline
- DestinationOrder
- Ownership
- DataRepresentation
- LatencyBudget
- Liveliness
- Partition
- Presentation

See Usage in the *RTI DDS Spy User's Manual* for more information.

3.7.6 New DDS Spy options for type inspection

This release introduces three new command-line options to *rtiddsspy* that allow you to inspect the type definition and the sample JSON schema of remote endpoints as they are discovered.

- `-showTypeDefinition`: Displays the type definition each time a new remote endpoint with type information is discovered.
- `-typeDefinitionFormat <XML|IDL>`: Specifies the format used to display the type definition when `-showTypeDefinition` is enabled. The default format is `IDL`.
- `-showSampleSchema`: Displays the sample JSON schema each time a new remote endpoint with type information is discovered.

The generated JSON schema is useful for understanding the structure of the data samples being received. These samples can be printed in JSON format using the `-printSample COMPACT` option.

Note: The generated JSON schema describes the structure of JSON-encoded data samples for a given type. It is not intended to be a complete or canonical representation of the corresponding IDL type. To obtain the full and authoritative IDL definition, use the `-showTypeDefinition` option.

See Usage in the *RTI DDS Spy User's Manual* for more information.

3.7.7 DDS Spy enhanced source locator display with IPv6 and shared memory support

DDS Spy now supports displaying IPv6 and shared memory source locators (the writer's locators for each received sample), in addition to the previously supported IPv4.

By default, only IPv4 locators are shown. You can customize which locators are displayed using the new `-srcLocators` option.

For example, specifying `-srcLocators "shmem|udpv4"` will show output like:

```
... from shmem(55161), 192.168.43.223: ...
```

See Usage in the *RTI DDS Spy User's Manual* for more information.

3.8 Deprecations and Removals

This section describes items that are deprecated or removed in 7.7.0, compared to 7.6.0. See also deprecations and removals in *Previous Releases*.

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

Any deprecations or removals noted in RTI's documentation 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. RTI's current standard terms and support and maintenance policies are available at https://www.rti.com/terms.

3.8.1 Python 3.8 and 3.9 no longer supported

See *Python API supports versions 3.10 to 3.14, no longer supports versions 3.8 and 3.9*.

3.9 Third-Party Software Changes

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

Table 3.1: Third-Party Software Upgrades

Third-Party Software	Old Version	New Version
Expat	2.7.1	2.7.3
Apache Log4j 2	2.17.1	2.25.3

The following third-party software is now used by *DDS Spy*:

Table 3.2: Third-Party Software Additions

Third-Party Software	Version
CivetWeb	1.16

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.

3.10 Other

3.10.1 Improved isolation for observability traffic

The configuration of the *DomainParticipant* created by *Monitoring Library 2.0* has been updated to improve traffic isolation.

By default, the *DomainParticipant* now runs in domain `101`, which is less commonly used than the previous default domain (`2`), reducing the likelihood of unintended interference with application traffic.

Additionally, the *DomainParticipant* is assigned the domain tag `RTI_o11y`, providing an extra layer of isolation for observability-related communication.

For more information, see Monitoring Library 2.0 in the *RTI Connext Observability Framework User's Manual*.

3.10.2 Aliases for enabled_transport QoS no longer require case-sensitive matching

The `enabled_transport` QoS required exact case matches for transport aliases. This issue has been resolved by making alias matching case-insensitive, allowing `enabled_transport` values to be accepted regardless of letter case.

3.10.3 Disallow unrealistic values of polling_period, to avoid errors collecting data for Observability Framework

Monitoring Library 2.0 uses the DDS_MonitoringPeriodicDistributionSettings::polling_period to periodically poll for new telemetry data to send to *RTI Observability Framework*. To avoid internal integer overflows, the `polling_period` must now be no more than 4294967 seconds (50 days). Enabling *Monitoring Library 2.0* will no longer succeed if the `polling_period` exceeds this value.

3.10.4 Allow disabling signal-blocking on Connext Threads for POSIX platforms

Historically, *Connext* has blocked signal handling in threads it creates and manages, for most signals and for POSIX platforms. This was done to ensure that the system is robust to unexpected signal handlers—for example, when a mutex is held.

To ensure that *Connext* can be used on platforms that rely on signal handling (for example, the Unity platform, whose GC (garbage collection) management relies on signal handling), it's now possible to disable signal-blocking by setting the environment variable `NDDS_DISABLE_SIGNAL_BLOCKING` to any value.

3.10.5 Convert DynamicType (or TypeCode) to JSON schema, for enhanced interoperability

This release adds an option to convert a DynamicType (or TypeCode) into a JSON schema. The generated JSON schema describes the structure of JSON data samples for a given type. This capability can be useful for sharing your data model with systems or tools that do not use DDS.

This option is supported in the following languages:

- C
- Traditional C++
- Modern C++

In C and traditional C++, a new value has been added to the `DDS_TypeCodePrintFormatKind` enumeration:

- `DDS_TYPE_CODE_PRINT_KIND_JSON_SCHEMA`

This output format can be used with the `print` and `to_string` APIs of `DDS_TypeCode`.

In modern C++, a new value has been added to the `DynamicTypePrintKind` enumeration:

- `json_schema`

This output format can be used with the `rti::core::xtypes::to_string` API.

See the DDS Consolidated JSON Syntax for more information.

3.10.6 Ensure compatibility with other DDS vendors using updated Compatibility.OtherDDSVendor.Enable QoS snippet

The **Compatibility.OtherDDSVendor.Enable** QoS Snippet has been modified to enable TypeObject v2 and disable the sending of TypeObject v1. This change ensures compatibility with systems that use TypeObject v2 (originally defined in the XTypes specification version 1.2) as the wire representation for types. By default, the receipt of TypeObject v1 information is still enabled, so applications using this snippet will still accept TypeObject v1 from applications that are not yet supporting TypeObject v2.

Additionally, this snippet now disables compression of TypeObject v1. Although the snippet does not send TypeObject v1, if you were to use the snippet and then explicitly enable TypeObject v1 in the same application (via modifying the QoS after the snippet was applied), the TypeObject v1 is no longer compressed. This change improves interoperability with vendors who don't support decompression.

See TypeLookup Service in the *RTI Connext Core Libraries User's Manual* for more information on TypeObject v1 and TypeObject v2. See QoS Profile Inheritance and Composition for more information on Snippets.

3.10.7 Added operation to retrieve remaining time until license expiration

A new operation has been added to the C and Modern C++ API's to get the remaining time (in seconds) until a license expires. This operation is only available in license-managed distributions. See Using an Activation Library in the *RTI Connext Installation Guide* for details.

Chapter 4

What's Fixed in 7.7.0

This section describes bugs fixed in *Connext* 7.7.0. These are fixes since 7.6.0.

Connext 7.7.0 is the first production release in the 7.7 long-term support (LTS) series. It is built upon and combines all of the fixes in releases 7.4.0, 7.5.0, and 7.6.0. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation.

[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.

4.1 Serialization and Deserialization

4.1.1 [Critical] Endpoint serialization/deserialization failure after deleting DomainParticipants if setting different xTypes compliance masks with NDDS_Config_set_xtypes_compliance_mask

Endpoints may have failed to serialize and deserialize data if you created *DomainParticipants* with different Extensible Types (xTypes) compliance masks (set with `NDDS_Config_set_xtypes_compliance_mask`) in the same application and then deleted some of those *DomainParticipants*. This resulted in a failure for existing *DataWriters* and *DataReaders* (on the still existing *DomainParticipants*) to serialize and deserialize data.

Now, you can use `NDDS_Config_set_xtypes_compliance_mask` to create and delete *DomainParticipants* with different xTypes compliance masks in the same application without affecting other *DomainParticipants*.

[RTI Issue ID CORE-16407]

4.2 Usability

4.2.1 [Minor] Lack of allowed multicast interfaces did not cause Connext to ignore multicast addresses

If none of your allowed interfaces supported multicast, then *Connext* failed to adapt to this situation and incorrectly did the following by default:

- Outgoing participant discovery announcements were still sent to the 239.255.0.1 multicast address.
- The 239.255.0.1 locator was still advertised in the outgoing participant discovery announcements.
- multicast_resend_threshold was still taking effect. Repair traffic (for both metadata and user data) was still attempted and failed to be sent over multicast.
- Warnings such as

```
NDDS_Transport_UDPv4_SocketFactory_create_receive_socket:No interface found␣
↪enabled for multicast
```

and

```
send failed: locator: udpv4://239.255.0.1:7400
```

were still being logged.

[RTI Issue ID CORE-14825]

4.2.2 [Minor] Error when configuring AllocationSettings with initial equals to maximum

Previously, when configuring `rti::core::AllocationSettings` with `initial_count` equal to `max_count` (to prevent dynamic allocations), *Connext* required `incremental_count` to be explicitly set to zero. This constraint caused errors during *DomainParticipant* creation when applied simultaneously to `remote_reader_allocation`, `local_reader_allocation`, and `local_writer_allocation`, due to internal allocation settings derived from these public configuration values:

```
ERROR [CREATE DP|0x00000002,0x00000002,0x00000002:0x000001C1{Domain=0}|CREATE␣
↪PRESPsService|LC:Discovery] REDAFastBufferPoolProperty_
↪isConsistent:PRECONDITION | propertyIn->growth.increment == 0 && propertyIn-
↪>growth.maximal (36) != propertyIn->growth.initial (12)
ERROR [CREATE DP|0x00000002,0x00000002,0x00000002:0x000001C1{Domain=0}|CREATE␣
↪PRESPsService|LC:Discovery] REDAFastBufferPool_newWithParams:!precondition:␣
↪inconsistent property
ERROR [CREATE DP|0x00000002,0x00000002,0x00000002:0x000001C1{Domain=0}|CREATE␣
↪PRESPsService|LC:Discovery] PRESPsService_new:!create␣
↪readerContentFilterPolicyBufferPool
```

This issue has been resolved by relaxing the validation requirements:

- When `initial_count` equals `max_count`, dynamic allocations will not occur regardless of the `incremental_count` value.

- The `incremental_count` value is now ignored in this scenario, eliminating the previous error.
- If `incremental_count` is explicitly set to zero, *Connext* will still validate that `initial_count` equals `max_count`.

[RTI Issue ID CORE-16228]

4.2.3 [Minor] Monitoring Library 2.0 required a license when used in DDS Ping or DDS Spy with license-managed bundles

When enabling *Monitoring Library 2.0* in *DDS Ping* or *DDS Spy* with License-Managed (`lm`) *Connext* bundles, a license is no longer required. (Note that *Collector Service* does still require a license.)

[RTI Issue ID MONITOR-756]

4.3 Transports

4.3.1 [Critical] Applications using Real-Time WAN may have stopped communicating after IP mobility events

Applications using the *Real-Time WAN Transport* could stop receiving data if multiple IP mobility events (for example, a network interface going down and then up) occurred close together in time. This issue could cause communication to stop after one of those events.

The issue only occurred when all of the following conditions were met:

- The platform was a Linux system.
- The application did not configure `comm_ports` in the *Real-Time WAN Transport* (that is, port assignment was automatic).
- Multiple IP mobility events occurred in close succession.

When this issue occurred, the following error messages were logged:

```
NDDS_Transport_UDP_WAN_CommPortsMappingTable_addMapping: !add a mapping for␣
↪the provided rtps port already exists
NDDS_Transport_UDP_create_recvresource_rrEA: !add comm ports mapping
DDS_DomainParticipantPresentation_update_participant_locatorsI: ERROR: Failed␣
↪to set default unicast entryports
```

[RTI Issue ID CORE-16439]

4.3.2 [Major] IP mobility functionality did not detect when network interfaces gained multicast capability

There was an issue preventing multicast communication when network interfaces gained multicast capability. *Connext* IP mobility functionality now properly detects when a network interface's multicast flag changes from disabled to enabled, and creates the required multicast sockets.

[RTI Issue ID CORE-13152]

4.3.3 [Minor] Zero-Copy DataReaders did not filter out redundant samples

Zero-Copy *DataReaders* had the property `dds.data_reader.state.filter_redundant_samples` set to false automatically and the property was unnecessarily sent in endpoint discovery samples. When this property is set to false, redundant samples coming from the same virtual GUID will not be filtered. Situations that would cause redundant samples include:

1. Redundant Routing Services forwarding the same samples from *DataWriter* to a *DataReader*.
2. Re-discovery after a liveliness loss event when using non-volatile durability.

In these scenarios, it is generally desired to only receive the samples a single time in an application. However, setting `dds.data_reader.state.filter_redundant_samples` to false would cause the application to receive the samples more than once.

The `dds.data_reader.state.filter_redundant_samples` property is no longer being set for Zero-Copy *DataReaders*. You can always choose to set this property to false in the *DataReader's* QoS if you would like to continue to not filter redundant samples.

[RTI Issue ID CORE-16181]

4.3.4 [Minor] Incorrect information in shared memory log message when out of resources

When the send operation of the shared memory transport failed due to the shared memory queue being full, the log message displayed the incorrect value for the `receive_buffer_size`, making the problem hard to debug. Now, the information in the message is correct and now also includes the `message_size_max`.

[RTI Issue ID CORE-15858]

4.3.5 [Minor] DomainParticipant failed to be enabled if a network interface with UDPv4 address 0.0.0.0 was detected

A network interface can temporarily have an address of 0.0.0.0 as a placeholder while waiting for DHCP to assign an address. If this happened, then enabling a *DomainParticipant* would have failed with the following error:

```
NDDS_Transport_UDPv4_SocketFactory_create_receive_socket:setsockopt(ADD_
↪MEMBERSHIP) error 0XF9
```

This problem has been fixed for the UDPv4 transport by ignoring any interface with address 0.0.0.0.

[RTI Issue ID CORE-15550]

4.4 Reliability Protocol and Wire Representation

4.4.1 [Critical] Sample may not have been delivered to a DataReader in a Required Subscription

A *DataWriter* configured with Required Subscriptions might not have delivered some samples to any *DataReader* that was part of those subscriptions. These samples were incorrectly lost.

This issue occurred when some samples had to be gapped (through GAP messages) to the *DataReaders* in the Required Subscription—for example, when a *DataReader* used a ContentFilteredTopic. It could also occur when *DataReaders* belonging to a Durable Subscription were restarted.

[RTI Issue ID CORE-16166]

4.4.2 [Critical] Samples lost if new DataReader created on same topic and DomainParticipant as existing DataReaders using async publishing

It was possible for existing *DataReaders* to incorrectly lose a few samples if a new *DataReader* was created in the same *DomainParticipant* and on the same topic as them.

This happened when the *DataWriter* writing the samples was using asynchronous publishing and the new *DataReader* was discovered after a sample was written by the *DataWriter* but before it was sent by the asynchronous publishing thread. If that happened, the samples pending to be sent by the asynchronous publishing thread were incorrectly announced as irrelevant to the existing *DataReaders* through a GAP submessage and those *DataReaders* never received the actual samples.

[RTI Issue ID CORE-16251]

4.4.3 [Critical] Sample may not have been delivered to a DataReader using application-level acknowledgments

A *DataWriter* might not have delivered some samples to any *DataReader* that was setting `reliability.acknowledgment_kind` to a value other than `PROTOCOL_ACKNOWLEDGMENT_MODE`. These samples were incorrectly lost.

This issue occurred when some samples had to be gapped (through GAP messages) to the *DataReader*—for example, when a *DataReader* used a ContentFilteredTopic. It could also occur when the *DataReader* was restarted.

[RTI Issue ID CORE-16421]

4.4.4 [Major] Unexpected precondition error sending repair to DataReaders with 16 locators

A *DataWriter* sending a repair to a *DataReader* with exactly 16 locators encountered a precondition error, such as the following:

```
SrWriterService.c:5315:RTI0x3000035:!precondition: "destEncapsulation-
Count >= ((2*(8)))"
```

This issue only occurred when using debug libraries.

[RTI Issue ID CORE-16471]

4.5 Performance and Scalability

4.5.1 [Critical] Performance degradation with durable writer history

In release 7.0.0, a significant performance degradation was introduced in *DataWriter* operations when the *DataWriter* was configured to use durable writer history.

This issue was caused by missing database indexes that should have been created to optimize performance. Without these indexes, certain database operations became substantially slower, affecting overall *DataWriter* throughput.

[RTI Issue ID CORE-16122]

4.5.2 [Critical] Performance degradation with Durable Writer History in write-intensive scenarios

In release 7.3.1, the throughput of write-intensive scenarios for a *DataWriter* configured with Durable Writer History was lower than in previous releases. This regression was caused by performance optimizations introduced in that version to improve read-intensive scenarios (for example, when handling late joiners)—see CORE-16122 above.

The fix for this issue improves the overall performance of write-intensive workloads. However, some degradation may still remain compared to earlier releases. If this issue affects your application, please contact support@rti.com for further assistance.

[RTI Issue ID CORE-16211]

4.5.3 [Major] Inefficient repair of fragmented samples caused performance degradation during large data transmission

When reliable, DDS-fragmented data needed to be repaired, only fragments from a single sample were requested per heartbeat. This led to significant performance degradation and slow recovery when transmitting large data, as multiple heartbeat round trips were required to repair multiple samples.

This issue has been resolved by allowing all missing fragments from all announced samples to be requested at the same time, greatly improving repair efficiency and large data transmission throughput in lossy network environments.

[RTI Issue ID CORE-14751]

4.6 Debuggability

4.6.1 [Minor] polling_period was not required to be a positive value

The *Core Libraries User's Manual* states that DDS_MonitoringPeriodicDistributionSettings::polling_period must be > 0 seconds. This requirement was not enforced. If the `polling_period` was not a positive value, then *Monitoring Library 2.0* was incorrectly allowed to be enabled.

[RTI Issue ID MONITOR-720]

4.6.2 [Minor] take_discovery_snapshot may have provided inaccurate information for remote endpoints

Using the `take_discovery_snapshot` APIs (for example, `rti::util::discovery::take_snapshot` in modern C++) could return inaccurate information for remote entities. Examples of incorrect data included:

- An incorrect entity name
- Incorrect partitions

[RTI Issue ID CORE-16182]

4.6.3 [Minor] Incorrect timestamp in log messages with microseconds higher than 500000

Since *Connext* 7.2.0, the function that printed the timestamp for log messages has been incorrectly rounding up the seconds value, if the microseconds were higher than 500000. This led to inconsistent timing in the logging showing the timestamp moving backwards:

```
[2026-02-11 14:27:19.951264] DEBUG rCo########Evt
  Mx06:RTIEventActiveGeneratorThread_loop:rCo########Evt sleeping {00000000,
↪2F80453E}
[2026-02-11 14:27:19.136920] DEBUG rCo########Evt
  Mx06:RTIEventActiveGeneratorThread_loop:rCoShant####Evt gathering events
```

This regression has been fixed, and now the timestamp is displayed properly.

[RTI Issue ID CORE-16508]

4.6.4 [Trivial] Backtraces did not correctly print addresses higher than 0xFFFFFFFF

Backtraces logged by *Connext* did not correctly print addresses higher than `0xFFFFFFFF`. Now, *Connext* prints backtraces showing the right addresses.

[RTI Issue ID CORE-16517]

4.7 TopicQueries

4.7.1 [Major] Inefficient O(n^2) reader-side performance when creating multiple TopicQueries simultaneously

There were several O(n^2) code paths that were executed when a *DataReader* with multiple outstanding TopicQueries received data. This has been resolved—the algorithms have been rewritten to O(n) complexity.

[RTI Issue ID CORE-16548]

4.8 Logging

4.8.1 [Trivial] Improved log messages for inconsistent WireProtocolQosPolicy values

The log messages that were printed out when there was an inconsistent QoS value in the WireProtocolQosPolicy were hard to understand. They have been improved to make it more clear exactly what value is inconsistent and what its value should be in order to be considered consistent. For example, previously, when the `WireProtocolQosPolicy::port_base` was set to a negative value, the log message used to be:

```
ERROR [CREATE DP|LC:Discovery] DDS_WireProtocolQosPolicy_is_consistentI:inconsistent QoS policy: port_base > 0
```

it is now:

```
ERROR [CREATE DP|LC:Discovery] DDS_WireProtocolQosPolicy_is_consistentI:INCONSISTENT QOS | port_base (-1) must be greater than 0.
```

[RTI Issue ID CORE-16034]

4.8.2 [Trivial] Confusing log message when DataWriter could not queue items *

When a *DataWriter* had a full queue or timed out when adding to it, the log message incorrectly warned about an OUT OF ORDER error instead of a lack of space.

[RTI Issue ID CORE-16143]

* *This bug does not affect you if you are upgrading from 7.3.x or earlier.*

4.9 APIs (C or Traditional C++)

4.9.1 [Minor] DDS_DataWriter_is_sample_app_acknowledged didn't check for NULL for is_app_ack parameter

`DDS_DataWriter_is_sample_app_acknowledged` did not check for `NULL` for the `is_app_ack` parameter.
This could have led to undefined behavior, such as crashes, if you passed in this invalid value.

[RTI Issue ID CORE-16367]

4.10 APIs (Modern C++ API)

4.10.1 [Critical] Implicit destruction of Modern C++ entities inside their Listeners

Some Modern C++ entities were being implicitly destroyed when they went out of scope within their Listeners, which is not permitted. Now, entities should no longer be unintentionally destroyed while their Listener is executing.

[RTI Issue ID CORE-15869]

4.10.2 [Major] Excessive memory allocation in rti::core::xtypes::convert function

The rti::core::xtypes::convert function allocated all bounded strings and sequence members in the returned object to their maximum size, even if the size of these members was far smaller than the maximum. Now, they are only allocated to the exact size that is required based on the input `DynamicData` object.

For example, consider the following IDL:

```
struct StructWithBigSequence {
    sequence<long, 2147483646> bigSequence;
};
```

When using the rti::core::xtypes::convert function to convert a `DynamicData` sample to a `StructWithBigSequence`, the convert function incorrectly allocated 2147483646 elements, even if the number of elements in the `DynamicData` sample was far fewer than that number. Now, the convert function only allocates the number of elements that are actually in the `DynamicData` sample.

[RTI Issue ID CORE-16310]

4.10.3 [Major] find_datawriters/readers APIs didn't work with XML-Based Application Creation

Previously, when a *DomainParticipant* was created from XML (`create_participant_from_config()`), its *DataReaders* and *DataWriters* had to be looked up with their explicit template type `T`. For that reason, methods such as `find_datareaders` and `find_datawriters`,* which returned `AnyDataReader` and `AnyDataWriter`, would fail. This issue has been resolved, and now it is possible to look up the type-erased `AnyDataReader` and `AnyDataWriter` objects.

However, the `AnyDataReader` and `AnyDataWriter` entities created in this scenario now have specific constraints. The following operations are not allowed on these entities until a typed `DataReader<T>`/`DataWriter<T>` has been constructed by calling `AnyDataReader.get<T>()`/`AnyDataWriter.get<T>()` at least once:

- Extension methods, via the operator `->`.
- The `close()` method.
- The `retain()` method.

`AnyDataReader` and `AnyDataWriter` entities created from a *DomainParticipant* that was not created using *RTI Connext XML-Based Application Creation* are not affected by these changes.

[RTI Issue ID CORE-8879]

4.10.4 [Minor] Incorrect implementation of BuiltinTopicReaderResourceLimits default constructor

The default constructor of `BuiltinTopicReaderResourceLimits` incorrectly set all of the fields to 0 or `false`. This problem did not affect the default constructor of the class that contains `BuiltinTopicReaderResourceLimits`, which is `DiscoveryConfig`. It only affected the construction of a `BuiltinTopicReaderResourceLimits` in isolation.

[RTI Issue ID CORE-16392]

4.10.5 [Minor] Missing Modern C++ Distributed Logger import library on Windows

In earlier versions of *Connext* for Windows (*Connext* 7.2.0 and later), the runtime library of Distributed Logger in Modern C++ (`rtidlcpp2.dll`) was shipped without its corresponding import library (`rtidlcpp2.lib`).

[RTI Issue ID DISTLOG-263]

4.11 APIs (Java)

4.11.1 [Major] Some BuiltinTopicKey_t's functions did not work properly

The following APIs in `BuiltinTopicKey_t` class always returned `true` instead of the correct return value:

- `is_unkeyed_reader`
- `is_unkeyed_writer`
- `is_participant`
- `is_user_entity`
- `is_builtin_entity`
- `is_vendor_entity`
- `is_vendor_builtin_entity`

Now they return the correct return value.

[RTI Issue ID CORE-16162]

4.12 APIs (C# API)

4.12.1 [Major] .NET API built with only .NET Standard 2.0 as target framework

RTI's .NET API was built with only .NET Standard 2.0 as the target framework, which may have caused some dependency issues in some .NET Standard 2.1 based projects. The .NET API now targets both standards, 2.0 and 2.1.

[RTI Issue ID CORE-13149]

4.12.2 [Minor] DomainParticipant creation may not have respected EntityFactory auto-enable policy in C# API

Creating a *DomainParticipant* with a `preEnableAction` argument didn't respect the EntityFactory auto-enable policy. The participant was enabled after the pre-enable action in all cases, even when `EntityFactory.AutoEnableCreatedEntities` was set to `false`. This problem has been resolved, ensuring consistent behavior with the EntityFactory QoS policy in this method.

[RTI Issue ID CORE-16155]

4.13 APIs (Python)

4.13.1 [Trivial] Python API: mypy failures on Windows due to non-ASCII character in docstrings

The Python API docstrings on Windows systems contained non-ASCII characters causing mypy to fail when reading `__init__.pyi` as UTF-8.

[RTI Issue ID PY-214]

4.14 Dynamic Data

4.14.1 [Minor] Unbounded Sequences incorrectly converted to string from DynamicType

Previously, when a `DynamicType` containing an *unbounded* sequence was converted to a string, it was incorrectly represented as a bounded sequence whose bound matched the internal maximum value. For example, an unbounded `sequence<long>` would be printed as `sequence<long,2147483647>`.

This behavior has been corrected. Now:

- Unbounded sequences are correctly converted to their unbounded string form, e.g. `sequence<long>`.
- Conversely, if a bounded sequence has a bound equal to or greater than the maximum unbounded limit, it will be treated as unbounded when converted to a string.

[RTI Issue ID CORE-16172]

4.15 XML Configuration

4.15.1 [Minor] Setting monitoring.telemetry_data.metrics in a derived profile or using an XML snippet did not override inherited values

When configuring QoS profiles in XML, setting the sequence `monitoring.telemetry_data.metrics` in a derived profile—either via a snippet or by defining the value directly—did not override the inherited value from the base profile or previously applied snippets. Instead, the previous sequence values remained in effect (merged behavior).

With this fix, the sequence is now always fully replaced when explicitly defined in a derived profile, whether through a snippet or directly in the profile.

For example:

```
<qos_library name="MyLibrary">
        <qos_profile name="MyBaseProfile">
            <participant_factory_qos>
                <monitoring>
                    <telemetry_data>
                        <metrics>
                            <element>
                                <resource_selection>//data_writers/*</
↪resource_selection>
                                <enabled_metrics_selection>
                                    <element>dds_data_writer_reliable_cache_*
↪</element>
                                </enabled_metrics_selection>
                            </element>
                        </metrics>
                    </telemetry_data>
                </monitoring>
            </participant_factory_qos>
        </qos_profile>
        <qos_profile name="MySnippet">
            <?rti-qos_snippet?>
            <participant_factory_qos>
                <monitoring>
                    <telemetry_data>
                        <metrics>
                            <element>
                                <resource_selection>//data_writers/MyWriter</
↪resource_selection>
                                <enabled_metrics_selection>
                                    <element>dds_data_writer_deadline_missed_
↪total</element>
                                </enabled_metrics_selection>
                            </element>
                        </metrics>
                    </telemetry_data>
                </monitoring>
            </participant_factory_qos>
```

```
        </qos_profile>
        <qos_profile name="DerivedProfileWithSnippet" base_name="MyBaseProfile
↪">
            <base_name>
                <element>MyLibrary::MySnippet</element>
            </base_name>
        </qos_profile>
        <qos_profile name="DerivedProfileDirect" base_name="MyBaseProfile">
            <participant_factory_qos>
                <monitoring>
                    <telemetry_data>
                        <metrics>
                            <element>
                                <resource_selection>//data_writers/MyWriter</
↪resource_selection>
                                <enabled_metrics_selection>
                                    <element>dds_data_writer_deadline_missed_
↪total</element>
                                </enabled_metrics_selection>
                            </element>
                        </metrics>
                    </telemetry_data>
                </monitoring>
            </participant_factory_qos>
        </qos_profile>
    </qos_library>
```

Before the fix, both `DerivedProfileWithSnippet` and `DerivedProfileDirect` contained **two entries** in the `<metrics>` sequence:

- `//data_writers/*`
- `//data_writers/MyWriter`

The explicitly defined sequence was merged with the inherited sequence instead of replacing it.

After the fix, both derived profiles now contain **only one entry** in the `<metrics>` sequence:

- `//data_writers/MyWriter`

Any explicitly defined `monitoring.telemetry_data.metrics` sequence—whether introduced via a snippet or directly in the derived profile—fully replaces the inherited values.

This behavior now aligns with *Connext's* documented rule that sequences defined in a derived QoS profile replace the corresponding sequences in the base profile.

[RTI Issue ID CORE-16518]

4.16 Entities

4.16.1 [Critical] TransportMulticast QoS locators not properly tracked by locator reachability *

A regression introduced in *Connext* 7.6.0 prevented locator reachability from tracking multicast locators configured in *DataReaders* using the TransportMulticast QoS policy. This issue prevented sending pings to this type of locator and could lead *Connext* to use unicast communication instead of multicast even if the communication through the multicast locator was working properly. Now the locators are properly tracked.

[RTI Issue ID CORE-16260]

4.16.2 [Minor] Misleading log messages when printing or converting to string non-default TransportMulticast QoS policy

Printing or using the `DDS_DataReaderQos_to_string` API on a *DataReader* QoS that had the TransportMulticast QoS policy set to a non-default value logged exception messages as if you were attempting to change the QoS after enabling the *DataReader*:

```
DDS_TransportMulticastQosPolicy_is_equalI:ERROR: Trying to change immutable␣
↪policy sequence length
DDS_DataReaderQos_equals_w_params: TransportMulticastQosPolicy unequal
```

Now the QoS is printed or converted into string without producing these messages.

[RTI Issue ID CORE-13427]

* *This bug does not affect you if you are upgrading from 7.3.x or earlier.*

4.17 Crashes

4.17.1 [Critical] Possible crash if configuration of Monitoring Library 2.0 was remotely modified while an entity was being destroyed

When using *Monitoring Library 2.0*, the metrics tracked can be remotely modified at runtime. If the change in the configuration happened at the same time that an entity was being deleted, and the entity was affected by the metrics that changed, a timing issue could trigger an update on the monitoring data from that entity, leading to a crash—for example, if *Monitoring Library 2.0* changed the metrics for all *DataWriters* in a *DomainParticipant*, and one of those *DataWriters* was being deleted. Now, the change of the metrics configuration is properly protected with respect to the deletion of entities.

[RTI Issue ID MONITOR-740]

4.17.2 [Critical] Crash when dividing by zero in XML type definition

When defining an invalid constant value in XML—for example, setting a default value to an invalid expression such as `42/0`—*Connext* crashed.

For example:

```
<dds>
    <type_library name="MyTypes">
          <const name="CONST_LONG" type="int32" value="42/0"/>
          <struct name= "MyStruct">
               <member name="m1" type="int32" default="CONST_LONG"/>
          </struct>
    </type_library>
</dds>
```

If you attempted to load this XML with:

```
dds::core::QosProvider qos_provider(xml_file_name);
```

the following error would occur:

```
DDS_ExpressionValue_divide:BAD PARAMETER FAILURE | 42LL division by zero.
```

[RTI Issue ID CORE-16126]

4.17.3 [Critical] DataWriter using Durable Writer History and not using Required Subscription crashed when communicating with a DataReader with a role_name

A *DataWriter* using Durable Writer History and not using Required Subscriptions crashed when trying to communicate with a *DataReader* that had configured a `role_name` in its ENTITY_NAME Qos. Now the communication happens properly between the *DataWriter* and *DataReader*.

[RTI Issue ID CORE-16285]

4.17.4 [Critical] Possible crash when deleting an Entity while Monitoring Library 2.0 was enabled

A possible crash or heap corruption error could occur if an *Entity* deletion failed while *Monitoring Library 2.0* was enabled. Subsequent deletion of the same *Entity* could trigger a double-free.

[RTI Issue ID MONITOR-763]

4.17.5 [Critical] Crash when concurrently modifying QoS profiles with Monitoring QoS policy set in default factory profile

There was a race condition that caused a heap-use-after-free error when multiple threads concurrently accessed or modified QoS profiles using `DomainParticipantFactory` or `QosProvider` APIs. This issue occurred only when the Monitoring QoS policy was explicitly set in the `DomainParticipantFactory` default QoS profile.

[RTI Issue ID CORE-16544]

4.17.6 [Critical] Crash when using DDS_RECOVER_INSTANCE_STATE_CONSISTENCY and multiple writers updating the same instances

A *DataReader* with **ReliabilityQosPolicy::instance_state_consistency_kind** set to RECOVER_INSTANCE_STATE_CONSISTENCY may have crashed after recovering liveliness with a *DataWriter* and then receiving samples from a different *DataWriter* for the same instances that the first *DataWriter* was writing.

[RTI Issue ID CORE-16604]

4.17.7 [Critical] Segmentation fault in DataReader that was created disabled when using instance state consistency

Consider the following scenario:

- A *DataReader* is created disabled (e.g., because its *Subscriber* sets `SubscriberQos.entity_factory.autoenable_created_entities` to false).
- The DataReader has set `DataReaderQos.reliability.instance_state_consistency_kind` to `RECOVER_INSTANCE_STATE_CONSISTENCY`.
- The *DataReader* is later enabled.
- The *DataReader* discovers and matches with a *DataWriter*.
- The *DataWriter* writes a sample.
- The *DataReader* stops matching with the *DataWriter* (e.g., because the *Publisher* has changed its `PartitionQosPolicy`).
- The *DataReader* matches with the *DataWriter* again.

The *DataReader* would experience a problem in the internal function `PRESInstanceStateRequestService_scheduleIstRequestOrDispose`. In release libraries, the *DataReader* would experience a NULL pointer dereference. In debug libraries, the *DataReader* would generate this error: `!precondition: "instanceStateRequestService == ((void *)0)"`.

[RTI Issue ID CORE-16586]

4.17.8 [Major] Precondition added to rti::flat::StringBuilder::set_string() and rti::flat::PrimitiveSequenceBuilder<T>::add_n()

A misuse of the APIs `rti::flat::StringBuilder::set_string()` or `rti::flat::PrimitiveSequenceBuilder<T>::add_n()` — specifically, passing `nullptr` to parameters that expect a valid pointer—can result in a crash.

It is the caller's responsibility to ensure that these parameters are not null.

When linking against debug libraries, this requirement is now enforced through precondition checks. However, in release mode these checks are omitted to keep FlatData as low-latency as possible, and therefore API misuse will not be detected at runtime.

[RTI Issue ID CORE-16134]

4.18 Hangs

4.18.1 [Critical] Possible hang in TCP transport when failing to send a message

When the TCP transport failed to send a message, it was possible for the transport to incorrectly hang after printing a log message containing the following text:

```
ERROR NDDS_Transport_TCPv4_Connection_blockingWrite:send failed
```

[RTI Issue ID COREPLG-769]

4.19 Memory Leaks/Growth

4.19.1 [Critical] Memory use by Monitoring Library 2.0 may have grown unbounded over time

The memory used by *Monitoring Library 2.0* could grow without bound over time. This behavior was more noticeable in highly dynamic systems where entities are frequently created and destroyed, or where liveliness conditions change continuously.

[RTI Issue ID MONITOR-751]

4.19.2 [Critical] Unbounded memory growth when setting receive_window_size on DataReader

There was a potential for unbounded memory growth under the following conditions:

- The application created at least one reliable *DataReader* with `reader_qos.protocol.rtps_reliable_reader.receive_window_size` set to a value larger than the default (256).
- *DataReader* instances with this configuration were created and deleted continuously.

Under these circumstances, memory usage could grow without bound. This issue was rare, and the memory increase occurred very slowly.

[RTI Issue ID CORE-16532]

4.19.3 [Critical] Memory leak in DataWriter when matching reliable DataReader

In *Connext* 6.1.0 a memory growth in a *DataWriter* when matching a reliable *DataReader*, which had been fixed via CORE-7652, was reintroduced. As a result, a *DataWriter* kept a small amount of memory around for a matching *DataReader*, even if the *DataReader* was deleted and unmatched. This may have led to unbounded memory growth over time as new *DataReaders* were created and destroyed. Notice that the leaked memory was returned to the heap after the *DataWriter* was deleted. Now the memory is managed properly and returned after the *DataReader* is unmatched.

[RTI Issue ID CORE-16299]

4.19.4 [Critical] History depth not honored after instance transitioned from NOT_ALIVE to ALIVE state

If an instance transitioned from a NOT_ALIVE state to the ALIVE state while there was a loan on the NOT_ALIVE sample, a mismatch occurred in the internal *DataReader* counters, making the *DataReader* not honor the History depth for that instance. If the History depth was set to 1 after the issue happened, there would be 2 samples allowed for that instance. Now the History depth is properly honored in this case.

[RTI Issue ID CORE-16369]

4.19.5 [Major] Memory leak when UDP receive socket creation failed

This issue was fixed in release 7.5.0, but not documented at that time.

If *Connext* failed to create a UDP socket for receiving messages on a specific port or multicast address, not all allocated memory was properly released, resulting in a memory leak.

Before the memory leak occurred, errors were logged in the `NDDS_Transport_UDP_create_recvresource_rrEA` function.

This issue affected any UDP transport plugin: UDPv4, UDPv6, or *RTI Real-Time WAN Transport*.

[RTI Issue ID CORE-16415]

4.19.6 [Major] Memory leak in Monitoring Library 2.0 when mixing library types

A memory leak occurred when enabling *Monitoring Library 2.0* in applications mixing different library types; for example, when linking statically to *Connext* libraries while attempting to dynamically load the *Monitoring Library 2.0*. In such scenarios, *Monitoring Library 2.0* activation would fail and leak the library's internal representation.

[RTI Issue ID MONITOR-754]

4.19.7 [Minor] Unbounded memory growth in Modern C++ when creating and deleting DynamicData topics of the same type

In Modern C++, there was an unbounded memory growth (which manifested as a `struct DDS_DynamicDataTypeSupport` in a heap snapshot) when creating and deleting `DynamicData` topics of the same type. For example, if you did the following:

```
DomainParticipant participant(domain_id());
StructType type("sometype");
type.add_member(Member("a", PrimitiveType<int32_t>()));
for (int i = 0; i < 10000; ++i) {
    dds::topic::Topic<DynamicData> topic(participant, "AppTopic", type);
}
rti::util::heap_monitoring::take_snapshot("myHeapSnapshot.txt");
```

you would incorrectly see 10000 `DDS_DynamicDataTypeSupport` structures in `myHeapSnapshot.txt`. The correct behavior is to see only one `DDS_DynamicDataTypeSupport` structure in the heap snapshot. This problem has been fixed in the Modern C++ API and may still exist in other language APIs.

[RTI Issue ID CORE-16007]

4.19.8 [Minor] Memory leak when disabling Monitoring Library 2.0 in Traditional C++

If *Monitoring Library 2.0* was enabled in a Traditional C++ application, you would have experienced a memory leak in the C function `DDS_DomainParticipantFactory_get_instance`. This leak only occurred when disabling *Monitoring Library 2.0* and did not lead to unbounded memory growth while *Monitoring Library 2.0* was enabled.

[RTI Issue ID MONITOR-757]

4.20 Data Corruption

4.20.1 [Critical] Data corruption in Monitoring Library 2.0 due to race condition during event and log processing

A race condition in *Monitoring Library 2.0* could lead to a use-after-free condition, resulting in data corruption.

Monitoring Library 2.0 processes events and log messages asynchronously using multiple internal threads. If a cleanup pass was attempted while another thread was busy processing events or log messages, the cleanup pass was silently skipped. Over time, this could leave stale references to memory that had already been freed.

Once a dangling reference existed, *Monitoring Library 2.0* could read or write corrupted data, leading to undefined behavior.

[RTI Issue ID MONITOR-765]

4.20.2 [Major] Monitoring Library 2.0 failed to create dependent Entities when type_object_max_serialized_length and type_code_max_serialized_length were 0 *

Monitoring Library 2.0 did not detect types when `type_object_max_serialized_length` and `type_code_max_serialized_length` were set to 0 in the DomainParticipantResourceLimits QoS policy. As a result, all dependent *Entities* (*Topics*, *DataReaders*, and *DataWriters*) were not created.

Monitoring Library 2.0 now correctly handles types when they are not configured for distribution, and all dependent *Entities* are created.

[RTI Issue ID MONITOR-744]

4.20.3 [Minor] Converting a GUID_t to a BuitinTopicKey_t had issues when high bit was set

This issue was fixed in release 7.6.0, but not documented at that time.

The `BuiltinTopicKey_t.from_guid()` method failed to convert properly if the incoming `GUID_t` had a byte pattern that set the highest bit in the resulting four-segment number in the conversion.

[RTI Issue ID CORE-15492]

* *This bug does not affect you if you are upgrading from 7.3.x or earlier.*

4.21 OMG Specification Compliance

4.21.1 [Major] Samples of mutable types may have been dropped when bit FLAG_MUST_UNDERSTAND was set on key members

Connext may have failed to deliver samples to applications, and reported them as lost, when receiving data on a *Topic* whose type:

- Has mutable extensibility.
- Derives from another type.
- Has at least one member marked as a key.

The failure would be triggered if the sending node set the bit FLAG_MUST_UNDERSTAND of the parameter IDs preceding each key member value in the serialized payload, as prescribed by the OMG 'Extensible and Dynamic Topic Types for DDS' specification.

This error may have only manifested in a cross-vendor interoperability scenario where *Connext* received data from a different vendor's DDS implementation. The error could also have been triggered if receiving data from a type with members marked as "must understand," for which the specification also requires the bit FLAG_MUST_UNDERSTAND be set in their serialized parameter IDs. Support for "must understand" members is introduced in this release. See Must-Understand Members in the *RTI Connext Core Libraries Extensible Types Guide* for more information.

Connext will now correctly deliver all samples on *Topics* whose type may have triggered this error.

Note: To maximize interoperability with previous *Connext* releases, *Connext* does not set FLAG_MUST_UNDERSTAND for key members of mutable types by default. You can change this default and force compliance with the specification by using the Extensible Types Compliance Mask.

[RTI Issue ID CORE-16465]

4.22 Interoperability

4.22.1 [Major] RTPS Header Extension timestamp used incorrect clock

The RTPS Header Extension timestamp was populated with the `internal_clock` instead of the `external_clock`. This could lead to the timestamp being populated with an invalid value (from the perspective of the remote host), if the `internal_clock` was set to the `monotonic` value. Timestamps should be generated with the `external_clock`. The RTPS Header Extension timestamp is now populated with the `external_clock`.

For more information about clocks in *Connext,* see Configuring the Clock per DomainParticipant.

[RTI Issue ID CORE-16250]

4.22.2 [Major] TypeConsistencyEnforcement::ignore_member_names did not work in some cases

Setting TypeConsistencyEnforcement::ignore_member_names to true did not work in some cases. These are some examples of the scenarios that have been fixed:

- Two mutable structures have a member with the same name but different ID and another member with the same name and ID:

```
@mutable
struct CORE16105Struct1 {
    long x1; //@ID 1
    long x5; //@ID 5
};
@mutable
struct CORE16105Struct2 {
    long x1; //@ID 2
    long x5; //@ID 5
};
DataWriter of type CORE16105Struct1 and DataReader of type CORE16105Struct2.
```

- Two unions have different member names but the same extensibility, number of members, and discriminator values for the same types:

```
union CORE16105Union1 switch(unsigned long) {
    case 1:
        short x1;
    case 2:
        long x2;
    case 3:
        long x3;
};
union CORE16105Union2 switch(unsigned long) {
    case 1:
        short x2;
    case 2:
        long x1;
    case 3:
        long x3;
};
DataWriter of type CORE16105Union1 and DataReader of type CORE16105Union2.
```

- Two arrays of arrays or two sequences of sequences have different member names but the same extensibility and dimensions (only XCDR was broken; XCDR2 was working):

```
@appendable
struct CORE16105Array20Uint32 {
    unsigned long x1[20];
};
@final
struct CORE16105Array10Array20Uint32 {
    CORE16105Array20Uint32 x1[10];
};
```

```
@appendable
struct CORE16105Array20Uint32Alt {
    unsigned long altx1[20];
};
@final
struct CORE16105Array10Array20Uint32Alt {
    CORE16105Array20Uint32Alt altx1[10];
};
DataWriter of type CORE16105Array10Array20Uint32 and DataReader of type␣
↪CORE16105Array10Array20Uint32Alt.
```

[RTI Issue ID CORE-16105]

4.23 Vulnerabilities

The following vulnerabilities are fixed in this release.

See also RTI Connext Security Bulletins and Advisories for a complete list of vulnerabilities in RTI releases that have been published through the CVE® Program. That list may be more up-to-date.

4.23.1 [Critical] Potential unauthorized access to instance information in Connext applications

An unauthorized access to instance information in *Connext* applications could have occurred while setting DDS_ReliabilityQosPolicy::instance_state_consistency_kind to DDS_RECOVER_INSTANCE_STATE_CONSISTENCY on a *DataReader*.

User Impact without Security

A vulnerability in *Connext* applications while enabling instance state consistency on a *DataReader* could have resulted in the following:

- Unauthorized access to limited information about instances (for example, how many alive and disposed instances a *DataWriter* has). This information does not include the serialized payloads of any samples.
- Remotely exploitable.
- Potential impact on the confidentiality, integrity, and availability of *Connext* applications.
- CVSS v3.1 Base Score: 8.6 High
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L
- CVSS v4.0 Base Score: 8.3 High
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:L/VA:L/SC:N/SI:N/SA:N

User Impact with Security

A vulnerability in *Connext* applications while enabling instance state consistency on a *DataReader* could have resulted in the following:

- Unauthorized access to limited information about instances (for example, how many alive and disposed instances a *DataWriter* has). This information does not include the serialized payloads of any samples.
- Remotely exploitable.
- Potential impact on the confidentiality, integrity, and availability of *Connext* applications.
- CVSS v3.1 Base Score: 6.4 Medium
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:L/A:L
- CVSS v4.0 Base Score: 6.1 Medium
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:H/AT:P/PR:L/UI:N/VC:H/VI:L/VA:L/SC:N/SI:N/SA:N

[RTI Issue ID CORE-16119]

4.23.2 [Critical] Potential Denial of Service in Connext applications when parsing XML files

Earlier releases of *Connext* depended on Expat 2.7.1, which is known to be affected by CVE-2025-59375, a publicly disclosed vulnerability. Release 7.7.0 uses a newer version of Expat that does not have this vulnerability. See the third-party software upgrades in What's New.

User Impact without Security

This vulnerability could have resulted in the following:

- Large dynamic memory allocations when parsing a maliciously crafted small document.
- Exploitable through a compromised local file system containing malicious XML/DTD files.
- Remotely exploitable through malicious RTPS messages.
- CVSS v3.1 Score: 7.5 High
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
- CVSS v4.0 Score: 8.7 High
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N

User Impact with Security

This vulnerability could have resulted in the following:

- Large dynamic memory allocations when parsing a maliciously crafted small document.
- Exploitable through a compromised local file system containing malicious XML/DTD files.
- CVSS v3.1 Score: 6.2 Medium
- CVSS v3.1 Vector:CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
- CVSS v4.0 Score: 6.9 Medium
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N

[RTI Issue ID CORE-16200]

4.23.3 [Critical] Potential heap buffer write overflow and memory leak in Connext applications when parsing an XML type

There was a possible heap buffer overwrite and memory leak when parsing an XML type.

User Impact without Security

This vulnerability affected the Core Libraries and all Infrastructure Services that rely on the Core Libraries.

- Buffer overflow when parsing a malicious XML file.
- Exploitable by providing malicious XML code to the applications during startup or runtime.
- *Routing Service* could be exploited by using a remote load administration command with malicious XML code.
- CVSS v3.1 Base Score: 8.2 HIGH
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:H
- CVSS v4.0 Base Score: 9.2 CRITICAL
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:L/VA:H/SC:N/SI:L/SA:H

User Impact with Security

Similar to “User Impact without Security”; Security reduces the Attack Vector by protecting the network.

- CVSS v3.1 Base Score: 6.8 MEDIUM
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:H
- CVSS v4.0 Base Score: 8.2 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:N/VI:L/VA:H/SC:N/SI:L/SA:H

[RTI Issue ID CORE-16507]

4.23.4 [Critical] Potential heap buffer read overflow in Connext applications when parsing an XML type

There was a possible heap buffer overread of one byte when parsing an XML type.

User Impact without Security

This vulnerability could have caused the following on any application using XML types:

- Heap buffer overread of one byte leading to a small impact on confidentiality and a low probability of a crash.
- Exploitable through malicious RTPS messages.
- Exploitable through a compromised local file system containing a malicious XML file.
- CVSS v3.1 Base Score: 6.5 MEDIUM
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L
- CVSS v4.0 Base Score: 6.3 MEDIUM
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:L/VI:N/VA:L/SC:N/SI:N/SA:N

User Impact with Security

This vulnerability could have caused the following on any application using XML types:

- Heap buffer overread of one byte leading to a small impact on confidentiality and a low probability of a crash.
- Exploitable through a compromised local file system containing a malicious XML file.
- CVSS v3.1 Base Score: 4.4 MEDIUM
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:L
- CVSS v4.0 Base Score: 4.8 MEDIUM
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:L/VI:N/VA:L/SC:N/SI:N/SA:N

[RTI Issue ID CORE-16494]

4.23.5 [Critical] Potential out-of-bounds write in Connext when using Durable Writer History

Connext may have written out of bounds when using Durable Writer History.

User Impact without Security

- Memory corruption leading to data corruption or a crash.
- Exploitable only by modifying the ODBC configuration files.
- CVSS v3.1 Base Score: 4.4 Medium
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:L
- CVSS v4.0 Base Score: 4.8 Medium
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:N/VI:L/VA:L/SC:N/SI:N/SA:N

User Impact with Security

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

[RTI Issue ID CORE-16252]

4.23.6 [Critical] Potential external entity attack (XXE) in Connext applications when parsing malicious XML files

The Core Libraries XML parser was not resistent against XML External Entity (XXE) attacks.

User Impact without Security

A vulnerability in *Connext* applications while parsing XML files could have resulted in the following:

- Potential impact on the confidentiality and availability of *Connext* applications. When exploited, the *Connext* application may have crashed or an attacker may have triggered unauthorized local file disclosure to access sensitive system data.
- Locally and remotely exploitable, by providing a malicious XML document.
- CVSS v3.1 Base Score: 9.1 CRITICAL
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H
- CVSS v4.0 Base Score: 8.8 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:H/SC:N/SI:N/SA:N

User Impact with Security

A vulnerability in *Connext* applications while parsing XML files could have resulted in the following:

- Potential impact on the confidentiality and availability of *Connext* applications. When exploited, the *Connext* application may have crashed or an attacker may have triggered unauthorized local file disclosure to access sensitive system data.
- Locally exploitable, by providing a malicious XML document.
- CVSS v3.1 Base Score: 7.1 HIGH
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:H
- CVSS v4.0 Base Score: 6.9 MEDIUM
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:H/SC:N/SI:N/SA:N

[RTI Issue ID CORE-16328]

4.23.7 [Critical] Potential heap-based buffer overflow when parsing malicious XML configuration files

Connext applications were vulnerable to a heap-based buffer overflow when parsing specially crafted XML documents. This vulnerability, tracked as CVE-2026-25210, existed in the `libexpat` third-party library. An attacker could exploit this by providing a malicious XML configuration file, potentially leading to an application crash, memory corruption, or arbitrary code execution. The issue was fixed by upgrading `libexpat` to version 2.7.4.

User Impact without Security

- Exploitable through a compromised local file system containing malicious XML files.
- Remotely exploitable through malicious RTPS messages.
- A heap-based buffer overflow occurred in the `doContent` function of `libexpat`. This memory corruption could lead to an application crash (Denial of Service) or potential integrity loss through arbitrary code execution with the privileges of the application process.
- CVSS v3.1 Base Score: 9.8 CRITICAL
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
- CVSS v4.0 Base Score: 9.2 CRITICAL
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

User Impact with Security

- Exploitable through a compromised local file system containing malicious XML files.
- While *Security Plugins* protect against network-based attacks (RTPS), they do not protect the local file system unless the file is a Governance or Permissions Document signed by a Permissions Authority. For standard QoS or configuration files, the impact remained a potential crash or integrity loss.
- CVSS v3.1 Base Score: 7.8 HIGH
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
- CVSS v4.0 Base Score: 8.5 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

[RTI Issue ID CORE-16539]

4.23.8 [Critical] Potential out-of-bounds read in Core Libraries on invalid XML constant expression

An out-of-bounds read could have happened when processing XML containing a constant whose expression value is ill-formed.

User Impact without Security

This vulnerability affected the Core Libraries.

- Out-of-bounds read when parsing a malicious XML file.
- Exploitable by providing malicious XML code to the applications during startup or runtime.
- CVSS v3.1 Base Score: 8.2 HIGH
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:H
- CVSS v4.0 Base Score: 9.2 CRITICAL
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:L/VA:H/SC:N/SI:L/SA:H

User Impact with Security

Similar to “User Impact without Security”; Security reduces the Attack Vector by protecting the network.

- CVSS v3.1 Base Score: 6.1 MEDIUM
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:H
- CVSS v4.0 Base Score: 8.2 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:L/VI:N/VA:H/SC:N/SI:L/SA:H

[RTI Issue ID CORE-16574]

4.24 Other

4.24.1 [Critical] AppAck messages exceeded underlying transport message size

A new property, `dds.data_reader.appack_keep_last_payload`, has been added. When enabled, the *DataReader* will only keep the last payload used to acknowledge a sample. This means that if multiple samples are acknowledged with different payloads, the *DataReader* will discard all acknowledged payloads except for the last one.

This is useful for reducing the size of AppAck messages and serves as a workaround for the known issue AppAck messages cannot be greater than underlying transport message size. The message size is reduced because an AppAck message contains a list of sequence number intervals. By discarding all payloads except the last one, *Connext* can collapse consecutive samples—acknowledged without a payload or with discarded payloads—into a single interval.

[RTI Issue ID CORE-16141]

4.24.2 [Trivial] Incorrect partition labels in DDS Spy

This issue was fixed in release 7.5.0, but not documented at that time.

DDS Spy displayed incorrect partition labels when a "New writer," "New reader," or "New data" was received. "New writer" and "New data" displayed an S (subscription) instead of a P (publication). "New Reader" displayed a P instead of an S.

[RTI Issue ID CORE-15125]

Chapter 5

Previous Releases

5.1 What's New in 7.6.0

This section describes what's new in the Core Libraries compared to release 7.5.0 (see *Previous Releases*).

RTI® Connext® 7.6.0 is an early access release. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation. Or start with the RTI Connext What's New for a launchpad to all products.

Note: For backward compatibility information between 7.6.0 and previous releases, see the Migration Guide on the RTI Community Portal.

5.1.1 Updated default heartbeat periods for faster repairs of lost samples

The out-of-the-box default values for the `RtpsReliableWriter::heartbeat_period`, `fast_heartbeat_period`, and `late_joiner_heartbeat_period` for *DataWriters* have been changed from 3 seconds to 200 milliseconds. The `RtpsReliableWriter::max_heartbeat_retries` QoS has also been changed from 10 to 150 in order to maintain the same amount of time it takes before a *DataReader* is considered inactive (30 seconds).

These changes affect only reliable *DataWriters*. The heartbeat period controls how fast samples can be repaired. Previously, user-created *DataWriters* sent heartbeats every 3 seconds by default; it therefore took up to 3 seconds before a *DataReader* requested a repair for a lost sample. The new defaults speed up the repair process. The faster heartbeat periods could mean that you may notice an increase in network traffic because there are more heartbeats and acknowledgments being sent. You may choose to slow down the defaults if your network is not lossy and you can wait longer for samples to be repaired if they are lost.

See DDS_RtpsReliableWriterProtocol_t, in the *RTI Connext Core Libraries User's Manual*, for more information.

5.1.2 DNS tracker now enabled by default to reduce chance of discovery problems when using hostnames in peer lists

Connext allows the use of hostnames instead of IP addresses when configuring peers for specific transports (e.g., UDPv4 and UDPv6). The DNS tracker keeps the IP addresses of these hostnames updated after *DomainParticipant* creation. The DNS tracker does this by creating a thread that regularly polls the DNS service. This thread detects changes in the IP address that a hostname is resolved to and updates the related peers accordingly.

Previous to this release, the DNS tracker was disabled by default and had to be enabled by setting the `DiscoveryConfigQosPolicy::dns_tracker_polling_period` to a finite value. Now, that QoS setting is set to 20 seconds out-of-the-box, so *Connext* will discover any changes in hostname IP address resolution by default. Otherwise, a participant will continue to reach out to the outdated IP address to try and discover other participants, potentially preventing discovery from ever occurring with those peers.

See Using DNS Tracker to Keep Peer List Updated, in the *RTI Connext Core Libraries User's Manual*, for more information.

5.1.3 Extensible Types compliance mask added to all APIs

The Extensible Types compliance mask and its corresponding APIs have been added to the Java, C#, and Python APIs, having previously been only in the C, Traditional C++, and Modern C++ APIs. See Extensible Types Compliance Mask in the *Connext Core Libraries Extensible Types Guide* for more information.

5.1.4 Updated code generation for 8-bit integer types

The code generation process for 8-bit integer types (octet, int8, and uint8) defined in IDL now distinguishes their internal representations within the TypeCode. While this update does not affect the functionality or usage of the generated code, we are providing this information for users who may observe these internal differences.

Specifically, in the Python API, the IDL octet type now generates a distinct type: `idl.octet`. This new type is functionally equivalent to `idl.uint8`. This change was necessary to accurately represent the different 8-bit integer types in internal structures such as the TypeCode.

5.1.5 Application no longer fails if platform does not support SOCKET_BUFFER_SIZE_OS_MAX

Attempts to use the `send_socket_buffer_size` SOCKET_BUFFER_SIZE_OS_MAX option (for example, NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX) on platforms that do not support it caused the application to fail.

Now, when you set the SOCKET_BUFFER_SIZE_OS_MAX option on an unsupported platform, the application will log an error message and fall back to using the default socket buffer size. This change ensures that platform-specific configurations do not disrupt execution or prevent configuration profiles from being shared across environments.

See Setting Builtin Transport Properties with the PropertyQosPolicy in the *RTI Connext Core Libraries User's Manual* for more information on the `send_socket_buffer_size` options.

5.1.6 Improved throughput when disable_positive_acks field in DataReaderProtocol QoS policy set to TRUE

This release improves application throughput when there are *DataReaders* configured with the `disable_positive_acks` field in the DataReaderProtocol QoS policy set to TRUE.

The performance improvement is more noticeable with transient local configurations when there are a large number of samples in the *DataWriter* queue.

5.1.7 New metadata fields for custom filters in modern C++, Java, and Python

This release introduces support for filtering based on the following metadata fields when developing custom filters in modern C++, Java, and Python:

- `sample_identity`
- `source_guid`
- `related_source_guid`
- `related_reader_guid`

See Custom Content Filters in the *RTI Connext Core Libraries User's Manual* for more information.

5.1.8 TCP Transport now uses POLL socket monitoring by default on Linux systems, offering significantly better responsiveness

This release changes the default socket monitoring mechanism for the TCP Transport on Linux architectures from `SELECT` to `POLL` . The new `POLL`-based socket monitoring (currently only supported for Linux architectures) offers significantly better responsiveness when handling socket events and raises the maximum number of sockets supported by TCP Transport on Linux architectures from `1024` to `32768` (see related fix COREPLG-756 in the Transports section of "What's Fixed in 7.6.0").

See `socket_monitoring_kind` in TCP/TLS Transport Properties, in the *RTI Connext Core Libraries User's Manual*, for more information.

5.1.9 Decreased bandwidth usage when not actively debugging remote systems

Previously when using the Remote Debugging feature in *Admin Console*, *Monitoring Library 2.0* sent all telemetry data generated by a *Connext* application to *Collector Service*. Now, telemetry data is sent only if an active *Admin Console* session requests the data, reducing unnecessary network traffic and improving system efficiency.

See Remote Debugging, in the *Admin Console User's Manual*, for more information.

5.1.10 New built-in profile makes it easier to monitor Connext applications across geographically separated networks

This release introduces a new built-in profile called `BuiltinQosLib::Generic.Monitoring2.WAN`, which allows connecting to a *Collector Service* instance over the WAN using the *RTI Real-Time WAN Transport*. Connecting over the WAN enables you to use *Observability Framework* to monitor and collect telemetry data from *Connext* applications that are distributed across geographically separated networks, even when they are behind NATs. The new profile makes it easier to set up the connection over the WAN.

For example:

```
<qos_library name="MyQosLibrary">
    <qos_profile name="MyApplicationProfile" is_default_participant_factory_
↪profile="true">
        <participant_factory_qos>
            <monitoring>
                <enable>true</enable>
                <distribution_settings>
                    <dedicated_participant>
                        <collector_initial_peers>
                            <element>udpv4_wan://38.21.45.34:4500</element>
                        </collector_initial_peers>
                        <participant_qos_profile_name>
                            BuiltinQosLib::Generic.Monitoring2.WAN
                        </participant_qos_profile_name>
                    </dedicated_participant>
                </distribution_settings>
            </monitoring>
        </participant_factory_qos>
    </qos_profile>
</qos_library>
```

For additional information, see Connecting to a Collector Service Over a WAN in the *Observability Framework User's Manual*.

5.1.11 Third-Party Software Changes

The following third-party software is now used by *Connext*:

Table 5.1: New Third-Party Software

Third-Party Software	Version
BearHttpsClient	0.2.8
json-builder	Commit ID fffd849

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

Table 5.2: Third-Party Software Upgrades

Third-Party Software	Old Version	New Version
Expat	2.6.3	2.7.1
SQLite	3.39.4	3.50.2

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.

5.2 What's Fixed in 7.6.0

This section describes bugs fixed in *Connext* 7.6.0. These are fixes since 7.5.0, described in *Previous Releases*.

Connext 7.6.0 is an early access release. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation.

[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.

5.2.1 Discovery

[Critical] Port collision on one channel of multichannel DataWriter caused discovery to fail even if another channel had no port collision

Consider the following scenario:

- You have a *DataReader*.
- You have a multichannel *DataWriter*, which uses specific ports 1234 and 5678.
- The *DataReader* discovers the *DataWriter*, but the matching process for one channel fails due to a port collision that generates this error: `assertRemoteWriter:!create entryPort`. Another process has already created a socket bound to port 1234.

The *DataReader* was incorrectly considered unmatched with the *DataWriter*, even though the other channel did not encounter a port collision.

[RTI Issue ID CORE-15842]

5.2.2 Serialization and Deserialization

[Critical] Unexpected error messages when disposing of an instance while using batching and setting serialize_key_with_dispose to TRUE

Consider the following *DataWriter* QoS:

```
<datawriter_qos>
     <batch>
          <enable>true</enable>
     </batch>
     <protocol>
          <serialize_key_with_dispose>true</serialize_key_with_dispose>
     </protocol>
</datawriter_qos>
```

The data type is keyed, and the serialized key is not aligned to a 4-byte boundary.

Imagine that DataWriter1 registers an instance, and then DataWriter2 uses the resulting InstanceHandle to write and dispose that instance. DataWriter2 successfully sends one batch with a data sample of a given instance and then attempts to send another batch whose first sample is a dispose message for that same instance.

When using release libraries, DataWriter2 would have successfully sent the second batch, but the *DataReader* that was reading the batch from DataWriter2 would have experienced this error:

```
ERROR MIGInterpreter_parse:submessage not aligned to 4
```

When using debug libraries, DataWriter2 would have failed to send the batch and generated this error:

```
mig.2.0/srcC/generator/Generator.c:1216:RTI0x200003a:!precondition: "!(sample␣
↪!= ((void *)0) && (((sample)->serializedData[encapsulationIndex].
↪serializedData.pointer == ((void *)0) || ((sample)->
↪serializedData[encapsulationIndex].serializedData.length & 0x3) == 0)) &&␣
↪(((sample)->protocolParameters.pointer == ((void *)0) || ((sample)->
↪protocolParameters.length & 0x3) == 0)))
```

This problem only affected 7.3.0 and above. DataWriter2 now successfully sends the batch, and the *DataReader* successfully receives the batch when using either release or debug libraries.

[RTI Issue ID CORE-15528]

5.2.3 Usability

[Critical] Locator Reachability DataReader could not receive assertions from more than four DomainParticipants

A misconfiguration in the Locator Reachability *DataReader* set the maximum number of instances to 4. As a result of this configuration, the Locator Reachability *DataReader* could not receive assertions from more than four *DomainParticipants*. Now, the Locator Reachability *DataReader* can receive assertions from as many *DomainParticipants* as configured in the `DomainParticipantResourceLimits.remote_participant_allocation` QoS setting.

[RTI Issue ID CORE-15703]

[Minor] DDS_RTPS_AUTO_ID_FROM_MAC failed if MAC address had all zeros for first 4 bytes or last 4 bytes

If you set the DomainParticipantQos `wire_protocol.rtps_auto_id_kind` to `DDS_RTPS_AUTO_ID_FROM_MAC`, and the hardware MAC address of your machine had four zeros at the start or the end, then *DomainParticipant* creation failed with this error:

```
PRESParticipant_new:!create facade
```

For example, a MAC address of `56,6f,c0,9e,0,0,0,0` or `0,0,0,0,56,6f,c0,9e` incorrectly caused *DomainParticipant* creation to fail.

Now, *DomainParticipant* creation succeeds regardless of the value of the MAC address.

[RTI Issue ID CORE-15655]

[Minor] Inconsistent QoS error when enabling Monitoring Library 2.0 with a default profile (is_default_qos = TRUE)

In earlier releases, enabling *Monitoring Library 2.0* by setting `<monitoring>/<enable>` to `TRUE` in `<participant_factory_qos>` could result in unexpected *inconsistent QoS policy* errors. These errors prevented *Monitoring Library 2.0* from being enabled.

This issue only occurred when the default profile—where `is_default_qos` was set to `TRUE`—contained QoS settings that were incompatible with the default `BuiltinQosLib::Generic.Monitoring2` profile, which is used to create the *Monitoring Library 2.0* entities.

[RTI Issue ID MONITOR-728]

5.2.4 Transports

[Major] TCP Transport did not properly enforce system FD_SETSIZE hard limits for select() socket monitoring

The TCP Transport did not properly enforce system `FD_SETSIZE` hard system limits for `SELECT` socket monitoring. This may have resulted in undefined behavior when using `SELECT` socket monitoring in systems where overwriting the `FD_SETSIZE` to a custom value at compilation time was not supported.

As a side-effect of this fix, the maximum `FD_SETSIZE` in Linux systems has been reduced to `1024`, which effectively limits both the maximum number of sockets (to `1024`) and the maximum socket ID (to `1023`) that are supported by `SELECT` socket monitoring. To avoid scalability issues in the Transport TCP in Linux systems, you are encouraged to switch to `POLL` socket monitoring, which is the new default (see "What's New in 7.6.0").

[RTI Issue ID COREPLG-756]

[Minor] Possible failure to create DomainParticipant if using multicast

If you were using multicast to receive either discovery or user traffic, it was possible for the ephemeral port used for sending unicast traffic to coincide with an intended multicast receive port, leading to *DomainParticipant* creation failure with this error:

```
DDS_DomainParticipant_enableI:Automatic participant index failed to␣
↪initialize. PLEASE VERIFY CONSISTENT TRANSPORT / DISCOVERY CONFIGURATION.
```

This problem has been solved by removing the possibility of a port collision, thereby removing that as a possible reason for *DomainParticipant* creation failure.

Note that a port collision is still possible when specifying a non-default value in the `multicast` or `unicast` fields within the `DataReaderQos` or `DataWriterQos` and creating a *DataReader* or *DataWriter*.

[RTI Issue ID CORE-15591]

5.2.5 Reliability Protocol and Wire Representation

[Critical] Unexpected data loss when using batching and finite reader resource limits

Consider the following scenario:

- A *DataWriter* enables batching.
- A reliable, keep-all history *DataReader* sets its `resource_limits.max_samples` to a finite value. For example, `max_samples` = 4.
- The *DataWriter* writes enough batches to exceed the `resource_limits.max_samples`, but the *DataReader* initially fails to receive one of them. For example, the reader receives batch sequence numbers 1, 3, 4, 5, and then batch sequence number 2 gets repaired.
- The application using the *DataReader* does not call `take()` to remove any of the received samples from the reader queue.

For the simplicity of this example, let's also assume that there is only one sample per batch.

The problem was that the *DataReader* would never deliver batches 2 through 4 to the user application. After fixing the problem, the *DataReader* now delivers batches 1 through 4 to the user application.

[RTI Issue ID CORE-16005]

5.2.6 Performance and Scalability

[Critical] Slower DDS Entity creation in large applications

In previous releases, enabling *Monitoring Library 2.0* significantly slowed down the creation of new DDS *Entities* in large applications with thousands of existing entities.

This issue was caused by a performance bottleneck when evaluating the `<resource_selection>` expression within the `<telemetry_data>` section of the *Monitoring Library 2.0* configuration.

This release partially addresses this issue by optimizing the evaluation of simple expressions such as `//*` and `//<resource_class>/*`. For example, using the resource selector `//data_writers/*` no longer results in a performance impact.

Future releases will include further improvements to fully resolve this problem.

[RTI Issue ID MONITOR-721]

5.2.7 Debuggability

[Major] Duplicate network traffic captures when creating WaitSets, AsyncWaitSets, or GuardConditions

Creating *WaitSets*, *AsyncWaitSets*, or *GuardConditions* could cause the same *DomainParticipantFactory* instance to be added multiple times to an internal global list of all the *DomainParticipantFactories* in an application. As a result, Network Capture (which uses the global list) may have captured traffic from the same *DomainParticipants* multiple times.

The *DomainParticipantFactory* instance is no longer added multiple times in this case and, consequently, Network Capture no longer captures *DomainParticipants*' traffic multiple times.

[RTI Issue ID CORE-15913]

5.2.8 Content Filters and Query Conditions

[Critical] Unexpected data loss when using writer-side content filtering and changing QoS

This issue was fixed in release 6.1.0, but not documented at that time.

Consider the following scenario:

- There are some *DataReaders* that are not using content filtering and other *DataReaders* that are using writer-side content filtering.
- There is a *DataWriter* whose QoS (or whose *Publisher*'s QoS) is incompatible with the *DataReaders*.
- At some point, the *DataWriter* (or *Publisher*) changes its QoS to be compatible with the *DataReaders*.
- Then, the *DataWriter* writes some samples.

None of the samples were sent to the *DataReaders* that did not use content filtering.

[RTI Issue ID CORE-15760]

[Critical] Historical sample loss for late-joining DataReaders setting protocol.virtual_guid

A late-joining *DataReader* setting `protocol.virtual_guid` may have failed to receive historical samples from a matching *DataWriter*.

This issue only occurred when *all* of the following conditions were met:

- `writer_qos.protocol.push_on_write` was set to TRUE (default) on the *DataWriter*.
- The *DomainParticipant* containing the late-joining *DataReader* also contained other *DataReaders* on the same *Topic* using ContentFilteredTopics.

[RTI Issue ID CORE-15992]

5.2.9 TopicQueries

[Major] max_samples_per_remote_writer resource limit not honored in some cases when using an unkeyed Topic and TopicQueries

If a *DataReader* created multiple TopicQueries and had a finite `max_samples_per_remote_writer` resource limit, that limit was not correctly enforced in TopicQuery queues. After some time, the *DataReader* may have only been able to accept fewer samples than `max_samples_per_remote_writer` before rejecting samples. This only happened if the *Topic* was unkeyed. Now, when a *DataReader* issues multiple TopicQueries, the `max_samples_per_remote_writer` resource limit is properly enforced for each TopicQuery queue.

[RTI Issue ID CORE-15639]

5.2.10 Logging

[Major] Unexpected warning during DomainParticipant enablement when locator reachability enabled

If you set the `DomainParticipantQos::discovery_config::locator_reachability_lease_duration` to a value other than `DDS_DURATION_INFINITE`, then *Connext* logged this warning during *DomainParticipant* enablement:

```
COMMENDAnonWriterService_assertRemoteReader:The remote reader with GUID␣
↪0x00000000,0x00000000,0x00000000:0x00020187 has no addressable locators.
```

This harmless warning was introduced in 7.0.0 and has now been removed.

[RTI Issue ID CORE-15718]

[Major] Misleading warning message when one non-loopback interface doesn't support multicast

If the network interfaces on a machine included an interface that supports multicast and a non-loopback interface that does not support multicast, and you sent a multicast packet from the machine, then a warning message like this one would incorrectly appear:

```
NDDS_Transport_UDP_send:FAILED TO SEND | Message with 64 bytes over 2␣
↪interface(s). Total bytes sent: 64. Total bytes to send: 128.
```

In this example, the multicast packet is 64 bytes, and *Connext* was incorrectly trying to send it over two interfaces instead of one (hence, the 128). This problem, which only affected versions 7.3.0 and above, has been fixed by only trying to send the packet over the interface that supports multicast.

[RTI Issue ID CORE-15833]

[Trivial] Isolated RTIOsapiUtility_strcat error on DomainParticipant creation

The creation of a *DomainParticipant* with several network interfaces and transports enabled could produce an error message similar to this:

```
ERROR [0x0101F566,0x1D4CA420,0x63609182:0x000001C1{Domain=0}
↪|ENABLE|LC:Discovery] RTIOsapiUtility_strcat:strcat(dest udpv4://192.168.1.
↪56:7412 udpv4://192.168.1.2:7412 udpv4://192.168.2.2:7412 udpv6://
↪FE80:0000:0000:0000:087F:2D88:7C58:91C3:7412 udpv6://
↪FE80:0000:0000:0000:8CDD:2AFF:FED2:283A:7412 udpv6://
↪FE80:0000:0000:0000:A4E1:A1D3:2686:A220:7412 udpv6://
↪FE80:0000:0000:0000:1507:6DBD:B23F:C38D:7412 udpv6://
↪FE80:0000:0000:0000:82BA:8AC8:9812:2319:7412 udpv6://
↪FE80:0000:0000:0000:CE81:0B1C:BD2C:069E:7412 udpv6://
↪FE80:0000:0000:0000:C889:F3FF:FECA:E764:7412 udpv6://
↪FDB2:2C26:F4E4:0000:0000:0000:0000:0001:7412 udpv6://FE80:0000:0000:0000:,␣
↪max size 526, src  ) returned NULL
```

This error message was harmless and could be ignored since it did not affect the *DomainParticipant's* behavior. The issue that was triggering the message has been fixed and the log message should not appear anymore.

[RTI Issue ID CORE-15866]

5.2.11 Dynamic Data

[Minor] Unable to set negative values for int8 union discriminators when using Dynamic Data

It was not possible to set negative values for a union discriminator with type int8 using the DynamicData API. There was an issue with how the type was being stored internally which caused the int8 value to be stored as an unsigned value instead of a signed value.

[RTI Issue ID CORE-14660]

5.2.12 APIs (C or Traditional C++)

[Minor] Traditional C++ DDS_KeyedString and DDS_KeyedOctets constructors allowed 0 for a string size

In the Traditional C++ API, the constructors for `DDS_KeyedString` and `DDS_KeyedOctets` incorrectly allowed you to pass in 0 for the size of a string. For example, calling

```
DDS_KeyedString *keyedString = new DDS_KeyedString(0 /* key_size */, 1 /*␣
↪size */);
```

incorrectly resulted in `keyedString->key` being non-NULL. This behavior was inconsistent with the C API, whose API `DDS_KeyedString_new_w_size` returns NULL if a string size is 0.

[RTI Issue ID CORE-15915]

5.2.13 APIs (Java)

[Critical] Java keyhash calculation error with mutable types in XCDR

A keyhash calculation error occurred in the Java implementation for mutable types using XCDR encoding, when one of the following conditions was met:

- The type's maximum serialized size exceeded 65 KB and the key was not primitive.
- At least one key member had a member ID greater than 16128.

Under these conditions, the keyhash was incorrectly computed, leading to interoperability issues between Java and other language implementations. Specifically, identical data instances produced by different writers were incorrectly treated as distinct, resulting in erroneous instance identification.

This fix ensures that Java now generates the same keyhash as the other language bindings, meaning that Java in 7.6.0 generates a different keyhash than it did in previous releases under the above conditions. If you want to preserve the previous behavior, see the Migration Guide for instructions on updating the Extensible Types compliance mask.

[RTI Issue ID CORE-15710]

5.2.14 APIs (Python)

[Major] Enhanced logging support in RTI Connext Python API with output_handler_extended

The RTI Connext Python API's `output_handler` implementation allows redirecting Connext log messages to a callable that receives the message as a plain string. This differs from other RTI Connext APIs, where the callables receive a `LogMessage` struct containing additional information such as log level, facility, and timestamp, not just the message text.

Previously, the Python API lacked this extended functionality. To address this, a new API called `output_handler_extended` was introduced. This API redirects log messages to a callable that receives

a `LogMessage` struct, providing all the informational fields available in other APIs. The `LogMessage` struct and its related types, such as `LogLevel` and `LogFacility`, were also added.

[RTI Issue ID PY-39]

5.2.15 APIs (Multiple Languages)

[Major] The result of cdr_serialized_sample_max_size was too small if the maximum size was greater than 4 GB

If a data type included sequences that caused its maximum serialized size to be greater than 4 GB, then the result of calling the TypeCode `cdr_serialized_sample_max_size` API was too small due to an integer overflow. This problem affected all language APIs and has been fixed by making this API return 2 GB under this circumstance.

[RTI Issue ID CORE-15726]

[Major] Non-Java register_type failed if the maximum size was greater than 2 GB

If a data type's maximum serialized size was greater than 2 GB, then the *DomainParticipant* `register_type` API failed with errors such as these:

```
RTIXCdrInterpreter_getSerSampleMaxSize:<type>:<field> skip error
RTIXCdrInterpreter_generateTypePluginProgram:failure generating get_max_
↪serialized_size program for type
DDS_DomainParticipant_register_type:!failed to register user type with␣
↪participant
```

This problem only affected *Connext* 6.0.0 and above and all language APIs except for Java. This problem has been fixed by making this API succeed under this circumstance. *Connext* supports registering types whose theoretical maximum serialized size is greater than 2 GB, as long as the samples that are written are never greater than 2 GB.

[RTI Issue ID CORE-12308]

[Minor] Enum label not printed when there was only one enum member

By default, when printing TypeCodes (using the `DDS_TypeCode_to_string` APIs), the ordinal values associated with an enum are only printed if they are explicitly provided in the type definition. There was a bug where the ordinal value was never printed (even if it was explicitly supplied in the type definition) if there was only one member in the enum.

[RTI Issue ID CORE-15715]

[Minor] Inconsistent QoS error when setting builtin discovery channels

In release 7.5.0, attempting to set `participant_qos.discovery_config.enabled_builtin_channels` to `DiscoveryConfigBuiltinChannelKind.All` in C# or `DiscoveryConfigBuiltinChannelKind.MASK_ALL` in Java during *DomainParticipant* creation resulted in an *inconsistent QoS* error.

[RTI Issue ID CORE-15868]

5.2.16 XML Configuration

[Trivial] Unexpected messages printed in console when there was an error parsing XML application configuration file

You may have seen unexpected messages printed in the console when there was an error parsing an XML application configuration file.

[RTI Issue ID CORE-15531]

[Major] Union types loaded from XML did not set ID values correctly when using autoid="hash"

Union types loaded from XML did not use the correct member ID values when using `autoid="hash"`. They instead defaulted to using sequential member IDs (as if using `autoid="sequential"`). Using `autoid="hash"` now correctly uses the hash of the field name as the member ID.

[RTI Issue ID CORE-11217]

[Major] XML namespace parsing error in QoS profiles

There was an XML parsing error when loading QoS profiles that include `xmlns` or `xsi:schemaLocation` attributes. These valid namespace declarations are now correctly accepted by the parser.

[RTI Issue ID CORE-15816]

5.2.17 Entities

[Major] Unable to create DistLogger singleton if remote administration was disabled

If you disabled remote administration using `RTI_DLOptions::setRemoteAdministrationEnabled(false)`, and then you attempted to create the DistLogger singleton using `RTI_DLDistLogger::getInstance()`, this attempt would have failed with these errors:

```
DL Error: RTI_DL_DistLogger_createInstance: Unable to set listener for␣
↪CommandRequests
DL Error: RTI_DistLogger_getInstance: Unable to create DistLogger singleton!
```

The problem affected all language APIs and has been fixed in all languages except Java; creating the DistLogger singleton now succeeds.

[RTI Issue ID DISTLOG-224]

5.2.18 Crashes

[Critical] Potential stack overflow in Connext applications when parsing XML files with a custom DTD with deeply nested entity references

The Core Libraries XML parser had a third-party dependency on Expat version 2.6.3, which is known to be affected by a publicly disclosed vulnerability. This vulnerability has been fixed by upgrading Expat to the latest stable version, 2.7.1. See the "What's New" section in this document for more details.

The impact on *Connext* applications of using the previous version was a stack overflow leading to a crash.

[RTI Issue ID CORE-15661]

[Critical] Possible crash when accessing information about a remote endpoint at the same time that the endpoint was being removed

The application could have crashed if it was trying to access information about a remote endpoint at the same time that the endpoint was being removed. (A remote endpoint is removed when the remote endpoint is deleted or loses liveliness.)

[RTI Issue ID CORE-15297]

[Major] Changes in RTI Connext logging behavior for Windows desktop applications

By default, Windows desktop applications (i.e., Windows applications with a graphical user interface) do not have an associated console for displaying *RTI Connext* log messages. To avoid losing important messages, *RTI Connext* automatically created a Windows console and redirected the output to it. The default behavior for Windows consoles attached to processes is to terminate the associated process when they are closed.

With *RTI Connext Express*, RTI's graphical tools (such as *Admin Console* or *Monitor* UI) displayed license-related messages (like trial end dates and trial limits) on a pop-up Windows console on startup. Closing that pop-up console caused the associated RTI tool to terminate unexpectedly. In *Admin Console*, a Java Virtual Machine error was displayed.

RTI Connext no longer creates Windows consoles for logging. RTI's graphical tools now display all messages (including license-related messages) in a mechanism specific to that tool (such as in a graphical view or log file). You should now install a logger device to handle your Windows desktop application logs. See Customizing the Handling of Generated Log Messages.

[RTI Issue ID CORE-15707]

[Critical] Modern C++ Distributed Logger may have crashed upon instance finalization

The Modern C++ Distributed Logger may have produced a crash after calling `DistLogger::finalize()` when a *DomainParticipant* had been set by the user. This problem was described as fixed in 7.3.0 (RTI Issue ID DISTLOG-238), but the fix was not complete.

[RTI Issue ID DISTLOG-255]

[Critical] Segmentation fault when using Network Capture APIs concurrently with DomainParticipantFactory deletion

Calling Network Capture APIs while another thread was deleting the `DomainParticipantFactory` could cause a segmentation fault due to unsafe concurrent access to internal resources. This issue has been resolved to prevent crashes in such scenarios.

[RTI Issue ID CORE-15914]

[Critical] Potential crash while printing the log message after an error

Due to mishandled string parameters on some log messages, an application running *Connext* could crash after an error had already happened. The problematic log messages are now properly generated and printed.

[RTI Issue ID CORE-15862]

[Critical] Segmentation fault when a DomainParticipant failed to be created

You may have seen a rare segmentation fault if a *DomainParticipant* failed to be created.

[RTI Issue ID CORE-16045]

[Critical] Segmentation fault and precondition errors with Monitoring Library 2.0 when Entities were deleted

Applications using *Monitoring Library 2.0* could experience a rare segmentation fault when *Connext* entities (such as *DataWriters*) were deleted. Additionally, when linking against the *Connext* debug libraries, you may have observed precondition errors in the log output.

[RTI Issue ID MONITOR-734]

[Critical] Possible crash when metric value changed when Monitoring Library 2.0 was being disabled

One of the *Monitoring Library 2.0* threads may not have finalized if a metric change happened at the same time that *Monitoring Library 2.0* was being disabled. That thread crashed after trying to access memory that was already freed. Now all threads are properly finalized.

[RTI Issue ID MONITOR-735]

5.2.19 Hangs

[Critical] Calling ignore_participant during a ParticipantBuiltinTopicDataDataReader on_data_available callback triggered by the deletion of a remote DomainParticipant led to a hang or unexpected error message

Suppose you were following the example code in Ignoring Specific Remote DomainParticipants for ignoring a *DomainParticipant* during a `ParticipantBuiltinTopicDataDataReader` `on_data_available` callback. When the callback was invoked as a result of deleting a *DomainParticipant,* it was possible for this `ignore_participant` to misbehave under either of the following conditions:

- You had called `set_listener` on the `ParticipantBuiltinTopicDataDataReader` after enabling the *DomainParticipant*.
- In a previous invocation of the `on_data_available` callback, you had not taken all of the samples, so in this invocation, there are multiple samples to take.

The misbehavior was as follows:

- In release libraries, `ignore_participant` would have succeeded, but a later call of `DDS_DomainParticipant_delete_contained_entities` would have hanged.
- In debug libraries, `ignore_participant` would have succeeded but with an error message in the internal function `REDACursor_startFnc`.

This problem has been fixed by making `ignore_participant` succeed without any side effects.

[RTI Issue ID CORE-15558]

[Critical] Potential deadlock when calling DDS_DomainParticipantFactory_set_qos and DDS_DomainParticipantFactory_finalize_instance

The application may have ended up in a deadlock in the following scenario:

- A custom XML file contained a structure.
- One thread called `DDS_DomainParticipantFactory_set_qos` while another thread called `DDS_DomainParticipantFactory_finalize_instance` at the same time.

[RTI Issue ID CORE-15541]

[Critical] Potential deadlock when using Network Capture utility APIs concurrently with DDS calls

Using the Network Capture utility APIs in one thread while making DDS calls in another thread could cause a deadlock during the first DDS API call in that thread.

The hang occurred because the creation of some DDS thread-specific state was taking a pair of locks in the opposite order in which they were taken by the Network Capture APIs.

[RTI Issue ID CORE-15984]

[Critical] Hang of QNX application using shared memory if one of the shared semaphores was in an unusable state

On QNX platforms, in the initialization of the shared memory transport, if a *Connext* application crashed or was killed between the creation and the initialization of a shared semaphore, that shared semaphore was unusable. Other *Connext* applications running on the same domain would try to access the shared semaphore and would hang waiting for the shared semaphore to be initialized. Now, the other *Connext* applications will log an error message and continue running if they wait for the shared semaphore initialization for more than four seconds.

[RTI Issue ID CORE-16108]

[Major] Calling ignore_participant when using the on_data_available callback to process a ParticipantBuiltinTopicData with partial_configuration led to a hang during DomainParticipant deletion

Suppose you were following the example code in Ignoring Specific Remote DomainParticipants for ignoring a *DomainParticipant* during a `ParticipantBuiltinTopicDataDataReader` `on_data_available` callback. When detecting the creation of another *DomainParticipant,* it was possible for this `ignore_participant` to misbehave when the ParticipantBuiltinTopicData::partial_configuration field was set to DDS_BOOLEAN_TRUE. The misbehavior was as follows:

- In release libraries, `ignore_participant` would have succeeded, but a later call of `DDS_DomainParticipant_delete_contained_entities` would have hanged.
- In debug libraries, `ignore_participant` would have succeeded but with an error message in the internal function `REDACursor_startFnc`.

This problem has been fixed by making `ignore_participant` succeed without any side effects.

[RTI Issue ID CORE-13783]

5.2.20 Memory Leaks/Growth

[Critical] Spike in memory usage when creating a DataWriter using Durable Writer History to restore samples

When creating a *DataWriter* that used Durable Writer History to restore a large number of samples, the memory usage incorrectly grew to a large amount and then shrunk back down. This problem has been fixed by reducing the memory usage.

[RTI Issue ID CORE-15802]

[Major] RTI DDS Spy showed DDS_DataReader_get_matched_publication_participant_data error before leaking memory

When discovering the deletion of a *DataWriter* or *DataReader* immediately after discovering the deletion of the *DomainParticipant* associated with the *DataWriter* or *DataReader*, *DDS Spy* incorrectly generated these errors before leaking memory:

```
DDS_DataReader_get_matched_publication_participant_data:ERROR: Failed to get␣
↪discovered_participant_data
get_matched_publication_participant_data error 4
ERROR DDS_SampleInfoSeq_set_maximum:failed to assert buffer must not be loaned
```

This problem only affected versions 7.3.0 and above.

[RTI Issue ID CORE-15629]

[Major] DomainParticipantFactory finalization failed even if all the DomainParticipants were deleted

When creating or deleting *DomainParticipants* concurrently from different threads, the finalization of the DomainParticipantFactory could fail, stating that "Not all the participants created were destroyed" even after deleting all the *DomainParticipants*.

[RTI Issue ID CORE-14609]

[Minor] Memory leak upon Transport TCP creation failure because of invalid configuration

A failure during Transport TCP creation may have triggered a memory leak. The leak triggered when providing an invalid PropertyQos configuration to the transport.

[RTI Issue ID COREPLG-755]

5.2.21 Data Corruption

[Critical] Serialization errors in Windows when publishing remote debugging telemetry with Monitoring Library 2.0

In release 7.5.0, you may have seen serialization errors like the following when publishing remote debugging telemetry with *Monitoring Library 2.0*:

```
DL Error: : ERROR [0x0101661B,0xF43B1B59,0x44F1F97A:0x00000283{Entity=DW,
↪Topic=DCPSEventStatusMonitoring,Type=DDS::Monitoring::Event,Domain=101}
↪|WRITE|ADD TO WRITER QUEUE|PROCESS SAMPLE] RTIXCdrInterpreter_
↪fastSerializeSample:DDS::Monitoring::TypeSupportQosPolicy:cdr_padding_kind␣
↪serialization error. Invalid enumerator value 533
DL Error: : ERROR [0x0101661B,0xF43B1B59,0x44F1F97A:0x00000283{Entity=DW,
↪Topic=DCPSEventStatusMonitoring,Type=DDS::Monitoring::Event,Domain=101}
↪|WRITE|ADD TO WRITER QUEUE|PROCESS SAMPLE] RTIXCdrInterpreter_
↪fullSerializeSample:DDS::Monitoring::DataReaderEvent:qos_type_support␣
↪serialization error
```

As a result, *Admin Console* may not have visualized all the entities in a remote *Connext* system.

This problem only affected Windows platforms and the following components:

- *Routing Service*
- *Persistence Service*
- *DataWriters* using zero copy transfer mode

[RTI Issue ID MONITOR-707]

5.2.22 OMG Specification Compliance

[Major] Incorrect value of an infinite duration in non-Java APIs

In all language APIs other than Java, the value of an infinite duration (e.g., dds::core::Duration::infinite() in Modern C++) was not compliant with the *OMG DDS Specification*. The incorrect value was `sec = 0x7fffffff`, `nanosec = 0xffffffff`. The correct value is `sec = 0x7fffffff`, `nanosec = 0x7fffffff`. This problem only affected releases 7.2.0 and above.

[RTI Issue ID CORE-15545]

[Minor] Consistency between max_samples_per_instance and max_samples was not enforced for unkeyed topics

The API Reference for max_samples_per_instance, which is a field that is included in the DDS Specification, states:

> While an unkeyed type is logically considered as a single instance, for unkeyed types this value has to be equal to max_samples or DDS_LENGTH_UNLIMITED.

This consistency was not enforced. For an unkeyed type, if `max_samples_per_instance` was less than `max_samples`, then entity creation would incorrectly succeed. Now, it fails with an error message prompting you to change the values to be equal.

[RTI Issue ID CORE-5643]

5.2.23 Vulnerabilities

The following vulnerabilities are fixed in this release.

See also RTI Connext Security Bulletins and Advisories for a complete list of vulnerabilities in RTI releases that have been published through the CVE® Program. That list may be more up-to-date.

[Critical] Potential heap buffer read overflow in Connext applications when using a malicious license string

An out-of-bounds read on the heap could occur while parsing a malicious license string property (e.g., `dds.license.license_string`) value.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Heap buffer overread while parsing a malicious license string.
- Exploitable by overwriting the XML QoS document on the file system with a malicious XML QoS document.
- Potential impact on confidentiality of *Connext* application.
- CVSS v3.1 Base Score: 4.4 MEDIUM
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:L
- CVSS v4.0 Base Score: 4.8 MEDIUM
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:L/VI:N/VA:L/SC:N/SI:N/SA:N

User Impact with Security

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

[RTI Issue ID CORE-15693]

[Critical] Potential invalid read memory access in Connext applications when subscribing to PublicationBuiltinTopicData

An invalid read memory access in *Connext* applications could have occurred after calling `DDS_Subscriber_lookup_datareader` to retrieve the builtin publication information and then discovering a *DataWriter*.

User Impact without Security

A vulnerability in *Connext* applications while discovering a *DataWriter* could have resulted in the following:

- Out-of-bounds read while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on the confidentiality of *Connext* applications.
- Potential crash in the application.
- CVSS v3.1 Base Score: 9.1 Critical
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H
- CVSS v4.0 Base Score: 8.3 High
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:N/VA:H/SC:N/SI:N/SA:N

User Impact with Security

There is no impact when enabling certain Security features; see Mitigations for more information.

Mitigations

- Use *Security Plugins* RTPS protection, discovery protection, or RTPS PSK protection.
- Set verbosity to `NDDS_CONFIG_LOG_VERBOSITY_WARNING` or higher for the `NDDS_CONFIG_LOG_CATEGORY_API` category.

[RTI Issue ID CORE-15730]

[Critical] Potential invalid read memory access in Connext applications during endpoint discovery

An invalid read memory access in *Connext* applications could have occurred while discovering a *DataWriter* or *DataReader*.

User Impact without Security

A vulnerability in *Connext* applications while discovering a *DataWriter* or *DataReader* could have resulted in the following:

- Out-of-bounds read while parsing a malicious RTPS message.
- Remotely exploitable.
- Potential impact on the confidentiality of *Connext* applications.
- Potential crash in the application.
- CVSS v3.1 Base Score: 9.1 Critical
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H
- CVSS v4.0 Base Score: 8.3 High
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:N/VA:H/SC:N/SI:N/SA:N

User Impact with Security

There is no impact when enabling certain Security features; see Mitigations for more information.

Mitigations

Use *Security Plugins* RTPS protection, discovery protection, or RTPS PSK protection.

[RTI Issue ID CORE-15789]

5.2.24 Other

[Critical] Possible Connext validation error while using a valid license

Some valid licenses may have failed *Connext* license validation.

[RTI Issue ID CORE-15738]

[Critical] Data races on architectures with weak memory models

On systems with weak memory architectures, such as Arm® and PowerPC®, data races could have led to undefined behavior or crashes in various *Connext* components.

[RTI Issue ID CORE-15638]

[Major] DomainParticipantFactoryQos could not be retrieved from QosProvider

There was no way to retrieve the DomainParticipantFactoryQos from the QosProvider. This functionality has now been added to the language bindings that did not previously support it: Modern C++, C#, and Python.

- Modern C++:

```
QosProvider qos_provider(user_qos_profiles_uri);
DomainParticipantFactoryQos dpf_qos = qos_provider->participant_factory_
↪qos();
// or qos_provider.extensions().participant_factory_qos();
```

- C#:

```
QosProvider qosProvider = new QosProvider(userQosProfilesUri);
DomainParticipantFactoryQos dpfQos = qosProvider.
↪GetDomainParticipantFactoryQos();
```

- Python:

```
qos_provider = dds.QosProvider(user_qos_profiles_uri)
dpf_qos = qos_provider.participant_factory_qos
```

[RTI Issue ID CORE-6845]

[Major] Incorrect hexadecimal formatting for uint8 and octet

Hexadecimal values in generated IDL were printed with an incorrect format (e.g., `<5a>`). The output has been updated to conform to the OMG Interface Definition Language™ specification, version 4.2, which requires that hexadecimal values start with the "0x" prefix (e.g., `0x5a`). This change ensures better compatibility and adherence to standards.

[RTI Issue ID CORE-15797]

[Major] Printing to XML missed quotes for external union members

When printing a union with `@external` members into XML format, the `external` attribute was missing quotes (e.g., `external=true` instead of `external="true"`), resulting in malformed XML that could not be parsed.

[RTI Issue ID CORE-15934]

[Major] Printing an IDL or XML enum with a default literal produced incorrect output

When printing an IDL or XML enum that included the `@default_literal` annotation in one of its enumerators, the output omitted this annotation. As a result, the generated type differed from the original definition.

[RTI Issue ID CORE-15931]

[Major] Printing an enum default literal printed the ordinal value instead of enumerator name

When printing in XML or IDL format, default values for enum types were incorrectly represented using the numeric ordinal value (e.g., `1`) instead of the enumerator name (e.g., `BLUE_1`). This resulted in invalid XML and IDL output.

Given this valid IDL:

```
enum ColorV1 {
    RED_1,
    @default_literal BLUE_1
};
typedef ColorV1 ColorV1Typedef;

struct ColorStructV1 {
    @key ColorV1Typedef color;
};
```

When printing the previous types, the resulting IDL and XML were:

```
// Idl
enum ColorV1 {
    @value(0) RED_1,
    @value(1) BLUE_1
};
@default(1) typedef ColorV1 ColorV1Typedef;

struct ColorStructV1 {
    @default(1) @key ColorV1Typedef color;
};

// Xml
<enum name="ColorV1">
    <enumerator name="RED_1" value="0"/>
    <enumerator name="BLUE_1" value="1"/>
```

```
</enum>

<typedef default="1" type="nonBasic" nonBasicTypeName="ColorV1" name=
↪"ColorV1Typedef"/>

<struct name="ColorStructV1">
    <member name="color" default="1" key="true" type="nonBasic"␣
↪nonBasicTypeName="ColorV1Typedef"/>
</struct>
```

This output is invalid, because the XML parser expects the enumerator name (`BLUE_1`) in the default attribute, not its numeric value.

The output logic has been updated so that when a default value refers to an enum literal, the enumerator name is printed in the XML and IDL representations.

[RTI Issue ID CORE-15930]

[Major] XML parser rejected external members with ranges excluding zero

The XML type parser failed to correctly handle external members with a specified min and max range that did not include 0, even though external members are not required to have a default value. This check caused the XML parser to reject otherwise valid type definitions.

Given this XML type definition:

```
<struct name="MyStruct" extensibility="final">
    <member name="myFloat" external="true" type="float32" min="10" max="11"/>
</struct>
```

Loading this type using the QosProvider would result in a failure, with the following error:

```
DDS_XMLTypeCode_getAnnotationParameterValues:Parse error at line 7:␣
↪annotation values are inconsistent. You must have min <= default <= max. If␣
↪min or max is specified and min > 0 or max < 0, you must specify the␣
↪default.
```

The validation logic has been updated, and now you can load external members with a min and max range that does not include 0.

[RTI Issue ID CORE-15929]

[Major] Incorrect string representation of union DynamicType with arrays of sequences

When printing (either in IDL or XML format) a union containing arrays of sequences, the array dimensions were missing from the output. This led to inaccurate or incomplete type representations.

Given the following IDL:

```
union ArraysOfSequencesUnion switch(long) {
case 0:
    sequence<short,2> myShortSequenceArray[2];
}
```

The previous (incorrect) string representations were:

```
// Idl
union ArraysOfSequencesUnion switch(long) {
    case 0:
        sequence<short,2> myShortSequenceArray;
}
// Xml
<union name="ArraysOfSequencesUnion">
    <discriminator type="int32"/>
    <case>
    <caseDiscriminator value="0"/>
        <member name="myShortSequenceArray" type="int16" sequenceMaxLength="2
↪"/>
    </case>
</union>
```

As shown, the array dimensions ([2]) were omitted from both outputs. The string representations now correctly include the array dimensions:

```
// Idl
union ArraysOfSequencesUnion switch(long) {
    case 0:
        sequence<short,2> myShortSequenceArray[2];
}
// Xml
<union name="ArraysOfSequencesUnion">
    <discriminator type="int32"/>
    <case>
    <caseDiscriminator value="0"/>
        <member name="myShortSequenceArray" type="int16" sequenceMaxLength="2
↪" arrayDimensions="2"/>
    </case>
</union>
```

[RTI Issue ID CORE-15928]

[Major] Error parsing an XML with an external typedef

If you tried to load the following XML:

```
<typedef type="nonBasic" nonBasicTypeName="MyStruct" name="MyStructTypedef"␣
↪external="true"/>
```

It complained with the following error: `RTIXMLParser_validateOnStartTag:Parse error at line 37: Unexpected attribute 'external'`. The above XML is now correctly parsed.

[RTI Issue ID CORE-15921]

[Major] Invalid escaping of special characters in default strings for XML and IDL output

When converting a DynamicType to a string in XML or IDL format, special characters in default string values (such as ", &, <, and >) were not properly escaped. This led to invalid XML and IDL outputs, breaking compatibility with parsers and tools—particularly during code generation workflows.

Default string values are now properly escaped according to the XML and IDL specifications:

[RTI Issue ID CORE-15886]

[Major] DynamicType with default string value incorrectly formatted for XML format

When converting a DynamicType to a string in XML format, if the DynamicType contained a default string, the output was previously `default=""default String""`. It now correctly renders as `default="default String"`.

[RTI Issue ID CORE-15847]

[Minor] Multiple attachments of same Condition to a WaitSet

Attaching the same `Condition` multiple times to a `WaitSet` did not report an error and allowed duplicate attachments. This led to inconsistent behavior: although each attachment appeared successful, calling `detach_condition` would only remove the condition once, leaving the `WaitSet` in an inconsistent state.

This issue has been resolved by updating `attach_condition` to perform a no-op and silently return success if the `Condition` is already attached. This ensures the internal state remains consistent without requiring additional checks in the application code.

[RTI Issue ID CORE-6857]

[Major] Error when parsing an XML with a struct that has an enum member with a default enumerator

If you tried to load the type `MyStruct` from the following xml:

```
<enum name="MyKind">
      <enumerator name="Kind0"/>
      <enumerator name="Kind1"/>
      <enumerator name="Kind2"/>
</enum>
<struct name= "MyStruct">
      <member name="an_enum_w_default" type="nonBasic"  nonBasicTypeName=
↪"MyKind" default="MyKind::Kind1"/>
</struct>
```

The XML parser in the Core Libraries complained because it was not able to find the enumerator name. The above XML is now correctly parsed.

[RTI Issue ID CORE-15919]

[Major] XML string literal interpretation divergence between Core Libraries and Code Generator

There was a mismatch in how XML string literals were interpreted between the Core Libraries and *Code Generator*:

- The Core Libraries interpreted string values literally.
- *Code Generator* treated unescaped strings as references, requiring values like "Foo" to be escaped as "Foo" to ensure they were parsed as literal values rather than references.

For example, to achieve the string `"Foo"`, the Core Libraries required `default="Foo"`, and *Code Generator* required `default=""Foo""`. This inconsistency led to unexpected or garbled values when using the same XML across different XML parsers.

This divergence has been resolved. Now, both the Core Libraries and *Code Generator* consistently interpret string constants. You can use `default="Foo"` in both, and it will correctly resolve to the string literal `"Foo"`.

[RTI Issue ID CORE-14664]

[Major] XML parser did not parse scientific notation when exponent included an explicit positive sign

The XML parser didn't correctly understand the literals expressed in scientific notation when the exponent included an explicit positive sign.

[RTI Issue ID CORE-15933]

[Major] Error when parsing an XML with a union that has a member with a default value

If you tried to load the type `MyUnion` from the following XML:

```
<union name="MyUnion" extensibility="mutable">
        <discriminator type="int16"/>
        <case>
        <caseDiscriminator value="1"/>
            <member name="member1" default="82" type="int16"/>
        </case>
</union>
```

It complained with the following error: `DDS_XMLTypeCode_getAnnotationParameterValue:Parse error at line 15: default annotation is not supported for optional members`. The above XML is now correctly parsed.

[RTI Issue ID CORE-15920]

[Minor] Building a Connext application with Wundef warning category could generate unexpected warnings

Building a *Connext* application with the Wundef category enabled could generate some warnings about undefined preprocessor directives being evaluated. The warning was harmless and had no implications in the final compiled code. The code has been enhanced to prevent the warning.

[RTI Issue ID CORE-15543]

5.3 What's New in 7.5.0

This section describes what's new in the Core Libraries compared to release 7.4.0 (see *Previous Releases*).

RTI® Connext® 7.5.0 is an early access release. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation. Or start with the RTI Connext What's New for a launchpad to all products.

Note: For backward compatibility information between 7.5.0 and previous releases, see the Migration Guide on the RTI Community Portal.

5.3.1 Python RPC API now fully productized and supported

This release promotes the Python RPC API from experimental to production-ready. It includes several improvements to the RPC and Request-Reply Python APIs that align the API with the future update of the OMG DDS-RPC standard and provide several new features, most notably an enhanced service discovery protocol.

The Python RPC and Request-Reply improvements in this release follow the *improvements for Modern C++ in 7.4.0* and bring the RPC/Request-reply APIs in these two languages up to par.

The enhanced service discovery protocol allows a Client (or a Requester) to reliably discover a Service (or a Replier) before making a call:

- Clients and Requesters can call a new method, `wait_for_service()`, to ensure that a service has been discovered.
- Clients and Requesters will fail immediately when making a request if no service has been discovered. This behavior can be disabled on the Client or Requester with the new argument `require_matching_service_on_send_request` on the Client and Requester constructor, for interoperability with request-reply APIs in other languages except Modern C++ and previous versions of the Python API.
- A Service or a Replier will only pass requests to the application layer when it can ensure that the reply can be delivered. This solves a previous race condition that could cause some initial replies for a Client or Requester to be lost.

Other changes include:

- Python code generation of interfaces is now supported by `rtiddsgen`, together with example generation when an IDL contains an interface and you use `-example universal`.
- The translation of service interfaces has changed to topic-types to align with a future update of the OMG RPC-DDS standard.
- Allows for the mutability of RPC operations by supporting the @final and @mutable annotations on a per-operation basis.

  For example, the following interfaces are compatible. A Foo client can call `my_method` on a Bar service, and the other way around (as long as they use the same service name):

```
@service("DDS")
interface Foo {
      @mutable
      long my_methodop(in short s, out double d);
};

@service("DDS")
interface Bar {
      @mutable
      void my_method(in short s, out double d, in long l);
};
```

- Allows the propagation of built-in exceptions `rti.rpc.RemoteUnsupportedOperationError`, `rti.rpc.RemoteInvalidArgumentError`, `rti.rpc.RemoteOutOfResourcesError`, `rti.rpc.RemoteUnknownOperationError`, and `rti.rpc.RemoteUnknownExceptionError`, in addition to the already-supported user-defined

exceptions. These built-in exceptions can be explicitly thrown by the service implementation. Additionally, `rti.rpc.RemoteUnknownOperationError` and `rti.rpc.RemoteUnknownExceptionError` are reported when the client calls an unknown operation in the service or the service implementation throws an unknown exception, respectively.

- Allows setting wire compatibility with `rti.connextdds.compliance.RpcMask` so that the `PID_RELATED_SAMPLE_IDENTITY RTPS` parameter is set to its standard value. By default, it continues to be set to a non-standard value for backward compatibility with previous *Connext* versions.
- Previously the Python API only allowed "in" parameters; support for "out" and "inout" parameters have been added allowing the modification of these parameters in the service side and propagating these changes to the actual parameters where the client invokes the operation.
- Added support for operation parameters of bounded string type.
- Added support for operation parameters of sequence type.
- Added support for @default, @min, and @max annotations on operation parameters.
- Added support for operation extensibility.
- Improved internal error management in the server. The server will notify the customer in case of an exception sending the response.

See Request-Reply Exchanges and Remote Procedure Calls (RPC) in the *RTI Connext Core Libraries User's Manual* for information on these APIs. Also see the remote_procedure_call and request_reply GitHub examples.

5.3.2 More efficient endpoint discovery for RPC and Request-Reply

This release provides faster discovery for RPC and Request-Reply entities. The `Requester.wait_for_service()` (for request-reply) or `Client.wait_for_service()` (for RPC) previously returned TRUE once the *DataWriter* and *DataReader* had both matched with a `Replier` (for request/reply) or `Service` (for RPC). However, this did not indicate that the external `Replier` or `Service` had also discovered the *DataWriter* and *DataReader*. This resulted in a need to send additional retries.

Now, `wait_for_service()` will not return TRUE until both the *DataWriter* and *DataReader* have matched **and** the *DataReader* has received a heartbeat from the `Replier's` or `Service's` *DataWriter*. This indicates that the remote *DataWriter* has completed endpoint discovery and is ready to write data. This should reduce the number of retries needed to send a request and have it be received.

This change is an improvement upon the Modern C++ API and is already included in the Python API as part of the new feature *Python RPC API now fully productized and supported*.

See Request-Reply Endpoint Discovery for more information on endpoint discovery for RPC and Request-Reply.

5.3.3 Improved DataWriter queue sample memory management eliminates required configuration for unbounded types

The way that *DataWriters* manage the memory that they use to store serialized samples has been improved to prioritize using pre-allocated memory. This enhancement eliminates the need to configure properties out-of-the-box for *DataWriters* that are using unbounded types.

Previously, *DataWriters* always tried to allocate buffers to the maximum serialized sample size of a type. For unbounded types, or even types with very large maximum serialized sample sizes, *DataWriter* creation would fail or consume large amounts of unused memory. To avoid this, the property `dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size` had to be set to a threshold value. Samples with serialized sizes below this value were stored in pre-allocated buffers and samples with serialized sizes above this value were stored in dynamically allocated buffers.

Now, *DataWriters* allocate buffers that are more closely sized to the sample sizes that are being written and keeps these buffers around so that they can be reused for subsequent samples that are similarly sized. This avoids dynamic memory allocation, which can hurt the performance and determinism of your application. No configuration is required to send large samples out-of-the-box.

As part of this work, the `dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size` property has been removed for all use cases except unbounded key members in Java. Setting it will have no effect, and a warning will be printed when a *DataWriter* that is using this property is created or has its QoS updated.

See Sample Memory Management for DataWriters in the *RTI Connext Core Libraries User's Manual* for a detailed explanation of how *DataWriters* manage the memory for serialized samples.

5.3.4 Enhanced out-of-the-box network performance with updated socket buffer sizes

Several improvements were made to enhance network performance by increasing the default socket buffer sizes and improving the usability of the socket buffer size configuration. These changes provide a better out-of-the-box experience while offering more flexibility in tuning the system for optimal performance.

As part of this update, this release has increased the default UDP socket buffer size from `131072` bytes to `2097152` bytes. This significant adjustment allows for more efficient data transmission and improved network throughput, addressing the demands of high-performance applications. To take full advantage of this updated default, you should adjust the kernel settings to match this new default value. See https://community.rti.com/kb/achieving-low-jitter-performance-connext-pro#Socket-buffer-sizes for details on how to do that.

In cases where the specified send or receive socket buffer size exceeds the maximum value allowed by the system, *Connext* will now automatically use the maximum value permitted by the system. A warning message (previously printed at local verbosity) is printed in such scenarios, providing suggestions to help you optimize your configuration and better understand system-level limitations.

To further improve usability, this release introduces new configuration options: `NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_MAX` (-2) and `NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_MAX` (-2). These options allow you to configure *Connext* to use the maximum socket buffer size allowed by the operating system. If the system does not define a maximum buffer size, *Connext* will fail and provide an error message indicating the issue. The previously available

configuration constants, `NDDS_TRANSPORT_UDPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT` (-1) and `NDDS_TRANSPORT_UDPV6_SOCKET_BUFFER_SIZE_OS_DEFAULT` (-1), remain unchanged and continue to allow the use of the OS default socket buffer size.

These enhancements provide greater flexibility and improve the usability of socket buffer size configuration while improving out-of-the-box network performance. By aligning *Connext* configuration with system-level constraints, these updates enable smoother operation and better overall efficiency.

For more information, see `send_socket_buffer_size` and `recv_socket_buffer_size` in Setting Builtin Transport Properties with the PropertyQosPolicy and Setting Real-Time WAN Transport Properties in the *RTI Connext Core Libraries User's Manual*.

5.3.5 Improved scalability for applications using writer-side content filtering

This release enhances the scalability of *Connext* applications by resolving issues related to writer-side content filtering. Previously, when two *DataWriters* within the same *DomainParticipant* performed writer-side filtering, they could interfere with each other, even if they were associated with different *Topics* and *Publishers*. Now, *DataWriters* in separate *Topics* and *Publishers* can perform writer-side filtering independently, without interference. This update significantly boosts concurrency, especially in multi-core systems.

5.3.6 Initial discovery performance improvements in large systems with participants containing many endpoints

This release has introduced performance improvements in large systems with *DomainParticipants* containing many endpoints by reducing the CPU required to match a remote endpoint (*DataReader* and *DataWriter*) to a local endpoint.

5.3.7 Eliminated requirement to manually clean up thread resources on platforms that support automatic thread-specific storage (TSS) cleanup

Whenever a *Connext* API is called by a user thread, some resources are stored in thread-specific storage. In previous *Connext* releases, the only way to clean up these resources was to called the `DomainParticipantFactory::unregister_thread` API (`ThreadContext::Dispose()` in the C# API) before a user-created thread exited.

This release adds support for automatic TSS cleanup on platforms that allow it. On platforms that support automatic TSS cleanup, it is no longer required to call the `DomainParticipantFactory::unregister_thread` API before a thread exits, although it is safe to do so.

Refer to the Platform Notes to see if your platform supports automatic TSS cleanup. (It will be mentioned in the "Thread Configuration" sub-section of the platform you are using.)

5.3.8 Improved flow control of fragmented sample repairs

When a *DataWriter* sends two fragmented samples (that is, their serialized size exceeds the configured `message_size_max` for the transport), and all fragments of the first sample are missed while only a few fragments of the second sample are missed, the *DataReader* will issue a `NACK` for the first sample and a `NACK_FRAG` for the second sample.

In such a scenario, the handling of fragmented sample repairs has been improved to ensure that the `max_bytes_per_nack_response` limit applies collectively to both `NACK` and `NACK_FRAG` repairs. Previously, this limit applied separately to `NACKs` and `NACK_FRAGs`, effectively doubling the configured `max_bytes_per_nack_response` limit. This enhancement ensures that when repairs are delayed using `min_nack_response_delay` and `max_nack_response_delay`, the limit is applied to all queued event repairs collectively. This allows for more efficient resource management and makes it easier to specify the maximum allowed size of a repair response.

Note: If repairs are not delayed, the limit still applies separately.

See DATA_WRITER_PROTOCOL QosPolicy (DDS Extension) in the *RTI Connext Core Libraries User's Manual* for more information.

5.3.9 RTPS messages containing a malformed inline Qos now trigger a warning, to avoid deserialization errors downstream

To improve the debuggability of *Connext* applications, this release introduces improvements to the RTPS Inline Qos parsing. Specifically, *Connext* can now detect the presence of an Inline Qos field that is located too early in the RTPS message. When this situation is detected, *Connext* logs a warning similar to the following one:

```
"COMMEND(Anon/Be/Sr)ReaderService_onSubmessage:skip error: protocol parameters
↪"
```

This improvement avoids cryptic deserialization errors (for example, but not limited to, a generic `available space` error) getting logged downstream.

5.3.10 Capture periodic DomainParticipant announcements in DISCOVERY logging category to enhance debugging of discovery

The `DISCOVERY` logging category has been enhanced to include periodic *DomainParticipant* announcements. This enhancement allows you to track the ongoing presence of discovered *DomainParticipants*, providing additional insights for debugging unexpected discovery behaviors. Previously, only the initial and final announcements were logged, while periodic announcements were not captured. With this enhancement, periodic announcements are now logged as they are sent.

5.3.11 Print incompatible QoS policies in a discovery snapshot for improved discovery debugging

The discovery snapshot API has been enhanced to print incompatible QoS policies. This enhancement allows you to identify the specific QoS policies responsible for the incompatibility between two entities when they fail to match. This enhancement improves the debugging process by presenting all mismatched QoS policies, enabling you to diagnose and resolve configuration issues more efficiently. See Discovery Snapshots in the *RTI Connext Core Libraries User's Manual* for more information.

5.3.12 Updated default value for socket_monitoring_kind to more performant WINDOWS_IOCP value on Windows systems

The default value for the `socket_monitoring_kind` property has been updated to `WINDOWS_IOCP` on Windows systems. This value was always recommended for better performance and scalability; now it is the default setting. See TCP/TLS Transport Properties in the *RTI Connext Core Libraries User's Manual*.

5.3.13 Support for new metadata fields in content filter expressions

This release introduces support for the following new metadata fields in content filter expressions in the SQL filter:

- `sample_identity.writer_guid.value`
- `sample_identity.sequence_number.low`
- `sample_identity.sequence_number.high`
- `source_guid.value`

For example:

```
@sample_identity.writer_guid.value = &hex(0708090A0B0C0D0E0F10111213141516)
```

See SQL Filter Expression Notation in the *RTI Connext Core Libraries User's Manual*.

5.3.14 InstanceHandle now provides a default constructor

The type dds::core::InstanceHandle was previously expected to be initialized with the special value `dds::core::InstanceHandle::nil()` or with another InstanceHandle. This caused unnecessary complexity, especially in the initialization of collections of InstanceHandles.

This release adds a default constructor that initializes an InstanceHandle to `nil()`.

5.3.15 Parameter type change to more convenient/appropriate size_t in pluggable transport's to_string APIs

The type of the `buffer_size_in` parameter has been updated from `RTI_INT32` to `size_t` in the following pluggable transport C APIs:

- `NDDS_Transport_Address_to_string`
- `NDDS_Transport_Address_to_string_with_class_id`
- `NDDS_Transport_Address_to_string_with_protocol_family_format`

For example, see NDDS_Transport_Address_to_string().

The behavior of the APIs has not changed, since the `buffer_size_in` parameter represents the size of the buffer used to get the address as a string, which should be positive. However, people using these functions could find benign warnings as a result of this change.

5.3.16 Deprecations and Removals

This section describes items that are deprecated or removed in 7.5.0, compared to 7.4.0.

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

Any deprecations or removals noted in RTI's documentation 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. RTI's current standard terms and support and maintenance policies are available at https://www.rti.com/terms.

"delete" method of DynamicDataTypeSupport deprecated

As a result of the fix for CORE-14359 (see *Memory Leaks/Growth*), the `delete` method of the `DynamicDataTypeSupport` class in the Java API has been deprecated. While its use won't trigger a compiler error, it will trigger a Deprecation warning and not do anything.

dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size removed (except for Java)

As part of the *new feature that eliminates required configuration for unbounded types*, the `dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size` property has been removed for all use cases except unbounded key members in Java. Setting the property will have no effect, and a warning will be printed when a *DataWriter* that is using this property is created or has its QoS updated.

The use of this property by Java is deprecated in this release and will be replaced by automatic behavior in an upcoming release.

5.3.17 Third-Party Software Changes

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

Table 5.3: Third-Party Software Upgrades

Third-Party Software	Old Version	New Version
Expat	2.6.2	2.6.3
LZ4	1.9.3	1.9.4

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.

5.4 What's Fixed in 7.5.0

This section describes bugs fixed in *Connext* 7.5.0. These are fixes since 7.4.0, described in *Previous Releases*.

Connext 7.5.0 is an early access release. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation.

[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.

5.4.1 Discovery

[Critical] Undefined behavior when serializing using any of the builtin discovery plugins

There was undefined behavior when serializing using any of the builtin discovery plugins. This issue could lead to crashes or memory corruption.

[RTI Issue ID CORE-15454]

[Critical] Incorrect locators used on newly created DataWriters or DataReaders after a Locator Reachability event

When creating a *DataWriter* or *DataReader*, the locators to be used for communication may be inherited from a local *DomainParticipant's* knowledge of the remote *DomainParticipant*. If a Locator Reachability event occurred before they were created, the *DataWriter* or *DataReader* might have attempted to use non-reachable locators.

[RTI Issue ID CORE-15390]

[Major] DomainParticipants using SPDP2 and fragmentation of the participant configuration messages failed to complete discovery

DomainParticipants using SPDP2 and fragmentation of the configuration messages (that is, the participant configuration message was larger than the `message_size_max`) failed to complete discovery, and the following warning was logged:

```
WARNING [PARSE MESSAGE|0x0101D6D6,0xC6933A70,0x66CB2656:0x00010187{Entity=DR,
↪MessageKind=DATA_FRAG}|RECEIVE FROM 0x0101D332,0x6CC2F8A4,
↪0x43BA1F0C:0x00010182] COMMENDSrReaderService_onSubmessage:!support␣
↪fragments. Message dropped.
```

Now *DomainParticipants* using SPDP2 can use fragmentation of the configuration messages.

[RTI Issue ID CORE-15257]

[Major] DomainParticipants failed to announce that endpoint was not reachable when internal clock was monotonic and external clock was real-time

DomainParticipants failed to announce to remote participants that an endpoint was not reachable at a locator when the internal clock was monotonic and the external clock was real-time.

[RTI Issue ID CORE-15366]

[Minor] DomainParticipants using LBED unregistered remote DataWriters when a remote participant was deleted instead of deleting them

DomainParticipants using LBED (and any participant discovery protocol) did not correctly remove remote *DataWriters* when a remote participant was deleted, but instead unregistered the remote *DataWriters*. This resulted in a NOT_ALIVE_NO_WRITERS sample on the builtin publication reader, rather than the expected DISPOSE sample. This did not result in a memory leak when using SPDP or SPDP2, since the remote *DataWriters* were then removed by the participant discovery plugin.

Remote *DataWriters* are now correctly removed when using LBED, and a DISPOSE sample is received on the builtin publication reader.

[RTI Issue ID CORE-15101]

5.4.2 Usability

[Minor] Unnecessary QoS consistency checks for best effort DataWriters

The `ResourceLimits::max_samples` QoS must be greater than or equal to the `RtpsProtocolReliableWriter::heartbeats_per_max_samples` and `RtpsProtocolReliableWriter::high_watermark` QoS values. This consistency check used to be made regardless of whether or not the *DataWriter* was using `RELIABLE Reliability::kind`. The `RtpsProtocolReliableWriter` QoS settings do not apply to best effort *DataWriters*, so the `max_samples` QoS is no

longer required to be consistent with any `RtpsProtocolReliableWriter` settings when the *DataWriter* is configured to use best effort communication.

[RTI Issue ID CORE-13685]

5.4.3 Transports

[Critical] Shared Memory communications might not have been recovered if a DataReader crashed or was ungracefully killed

If a *DataWriter* communicated with a *DataReader* using the Shared Memory Transport plugin, and the *DataReader* crashed or was ungracefully terminated, the *DataWriter* might have failed to communicate with a new *DataReader* spawned on the same machine.

Additionally, if multiple *DataReaders* were present on the same machine and one of them crashed or was ungracefully terminated, communication between the *DataWriter* and the remaining *DataReaders* might have been disrupted.

This issue has been resolved. Note that for this problem to occur, the crash or termination had to happen at a very specific moment, making it difficult to reproduce.

[RTI Issue ID CORE-15247]

5.4.4 Reliability Protocol and Wire Representation

[Critical] Writer-side filtered samples not always marked as acknowledged when application acknowledgment was used

When application acknowledgments were used in conjunction with ContentFilteredTopics, samples that were writer-side filtered were not always immediately marked as acknowledged. This problem, which was documented as fixed via RTI Issue ID CORE-6132 in previous releases (such as 7.3.0), was not completely fixed in those releases. Specifically, *DataWriters* failed to acknowledge writer-side filtered samples when the *DataReader* used a filter expression that was based only on key fields or only on metadata fields. This problem is now fixed, and *DataWriters* now always mark writer-side filtered samples as acknowledged.

[RTI Issue ID CORE-14980]

5.4.5 Performance and Scalability

[Critical] High CPU usage and delayed events in a DataReader managing many instances of a deleted DataWriter

When a *DataWriter* is deleted, all matching *DataReaders* must update all instances the deleted *DataWriter* was writing to indicate there was one less writer for that instance. The search through the instance list in the *DataReader* was not optimized to find the instances for a single *DataWriter*, so the deletion of a *DataWriter* could take a long time if the number of instances in the *DataReader* queue was large (10s or 100s of thousands). This processing could have delayed other important middleware events from running, like liveliness assertions and data processing.

Now, the *DataReader* keeps track of the instances that a *DataWriter* is writing so that it does not search through its entire list of instances. This change resolves the issue when the deleted *DataWriter* is writing a small subset of those instances. If the number of instances that a single *DataWriter* is writing is still large, the problem described will still exist.

[RTI Issue ID CORE-14906]

5.4.6 Debuggability

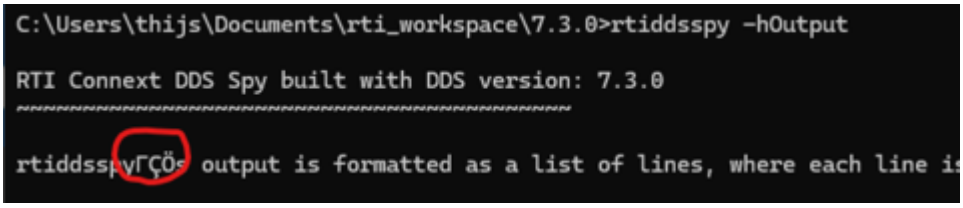
[Major] Incorrect source port for some Real-Time WAN Transport packets recorded with network capture

The source port for *RTI Real-Time WAN Transport's* bind-ping and discovery traffic packets was incorrect in network traffic captures taken with Network Capture. Instead of the correct port, a random integer was recorded.

[RTI Issue ID CORE-15462]

[Trivial] RTI DDS Spy showed unprintable character in help output

When printing the help output (using `rtiddsspy -hOutput`), *DDS Spy* showed an unprintable character:



This character has been fixed, and now the help output is printed as expected.

[RTI Issue ID CORE-15093]

5.4.7 Content Filters and Query Conditions

[Minor] Setting parameters on QueryCondition could have led to slightly inefficient refiltering

When setting the parameters on a QueryCondition, a buffer may have been allocated that was not needed. This caused the code to be slightly less efficient.

[RTI Issue ID CORE-15022]

5.4.8 Request-Reply and Remote Procedure Calls

[Minor] Request-reply and RPC readers created without explicitly loading BuiltinQosLib::Pattern.RPC did not set reliability.max_blocking_time to 10 seconds

Request-reply and RPC readers are configured to load a default set of QoS policies, regardless of whether `BuiltinQosLib::Pattern.RPC` is loaded as the QoS profile. These QoS policies match those defined in `BuiltinQosLib::Pattern.RPC`. However, readers created using the Python API did not set the `reliability.max_blocking_time` to the intended value of 10 seconds, and instead left it at the default value of 100 milliseconds. This problem affected only the Python API. The problem has been resolved; now, request-reply and RPC readers set the default `reliability.max_blocking_time` to 10 seconds even when `BuiltinQosLib::Pattern.Rpc` is not set as the QoS profile.

[RTI Issue ID REQREPLY-205]

5.4.9 Logging

[Trivial] Incorrect informational log messages when adding or removing remote writers in local readers

Some log messages, logged at a REMOTE logging verbosity, incorrectly logged that remote writers were being asserted or removed by local writers. For example:

```
REMOTE [0x0101CD1D,0x9B43DB76,0x623756A2:0x00000000|ASSERT REMOTE␣
↪DW|:0x000200C7{Entity=DR,Topic=PRESInterParticipantTopic,
↪Type=PRESInterParticipantParameter,Domain=0}|LINK 0x0101465D,0x21EAFEE8,
↪0x7AB36BA6:0x000200C2{Type=PRESInterParticipantParameter}|LC:Discovery]␣
↪PRESPsService_assertRemoteSessionWriterInLocalReader:assert remote writer:␣
↪0x0101465D,0x21EAFEE8,0x7AB36BA6,0x000200C2, in local writer: 0x000200C7␣
↪with best effort reader service
...
REMOTE [0x0101CD1D,0x9B43DB76,0x623756A2:0x00000000|REMOVE REMOTE DW␣
↪0x0101465D,0x21EAFEE8,0x7AB36BA6:0x000100C2|REMOVE REMOTE DW|:0x000100C7
↪{Domain=0}] PRESPsService_removeRemoteSessionWriterFromLocalReader:remove␣
↪remote writer: 0x0101465D,0x21EAFEE8,0x7AB36BA6,0x000100C2, from local␣
↪writer 0x000100C7 with best effort reader service
```

The log messages have been updated to say 'local reader' instead of 'local writer' in these cases.

[RTI Issue ID CORE-15417]

5.4.10 APIs (C or Traditional C++)

[Minor] Incorrect return code for DomainParticipant::assert_liveliness API on disabled participant

When the `DDS_DomainParticipant_assert_liveliness()` function in the C API was called on a disabled *DomainParticipant*, the function returned `DDS_RETCODE_BAD_PARAMETER`. The return code has been changed to `DDS_RETCODE_NOT_ENABLED`.

[RTI Issue ID CORE-12191]

5.4.11 APIs (Java)

[Major] Unnecessary memory operations when receiving inherited bounded sequences

Memory usage related to receiving samples containing bounded sequences in the Java API had been recently improved via CORE-14667. However, those initial changes did not correctly cover the case of inherited sequences. Now they do.

[RTI Issue ID CORE-15092]

[Major] For types with strings, sample size calculation in Java may have overestimated serialized size

In Java implementations, sample size calculations for types with strings assumed the maximum UTF-8 character size (4 bytes), potentially overestimating memory requirements. This behavior has now been optimized to calculate size based on the actual string content, reducing unnecessary overhead.

[RTI Issue ID CORE-15311]

5.4.12 APIs (C# API)

[Trivial] .NET bundle information not displayed in Launcher

Due to an error in the package definition, the .NET bundle information was not displayed in *RTI Launcher* once the package was installed.

[RTI Issue ID INSTALL-1033]

5.4.13 APIs (Python)

[Critical] Incompatibility between Python API and other APIs when using int8 type

Communication between Python and the other supported language APIs did not occur when using an int8 type. The int8 types were incorrectly treated as characters instead of octets, leading to communication failures.

[RTI Issue ID PY-190]

[Major] Python API didn't support reading TopicQuery samples exclusively

The Python API didn't allow creating a `ReadCondition` with the option to filter samples received from a TopicQuery. (It was possible to read all samples, both "live" and from a TopicQuery at the same time.)

The expected way to do that is to create a `ReadCondition` as follows, after creating a TopicQuery, and then use that condition in a `WaitSet` and a `reader.select().condition()` statement.

```
condition = dds.ReadCondition(
    reader, dds.DataStateEx(stream_kind=dds.StreamKind.TOPIC_QUERY)
)
```

However, the creation of the condition above failed with an exception that didn't allow the usage of a `DataStateEx` input argument.

Now the code above is valid.

[RTI Issue ID PY-188]

[Minor] Python API didn't provide enumeration to set Real-Time WAN Transport in code

The Python API didn't provide the enumeration `dds.TransportBuiltinMask.UDPv4_WAN` to enable the *Real-Time WAN Transport* in `dds.DomainParticipantQos.transport_builtin.mask`.

Enabling it via XML configuration was allowed.

The enumeration is now available.

[RTI Issue ID PY-191]

5.4.14 APIs (Multiple Languages)

[Minor] Some properties no longer accepted LENGTH_UNLIMITED string as a valid value

Some properties, such as `dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size`, could not be set to the special "unlimited" value using the string "LENGTH_UNLIMITED". You could have used "-1" instead. The properties now accept LENGTH_UNLIMITED.

[RTI Issue ID CORE-14328]

[Minor] Request-reply and RPC may have used standard PID instead of selected one

Request-reply and RPC should define the PID_RELATED_SAMPLE_IDENTITY (standard or legacy) based on the environment variable, the QoS definition, or the selection made with the API (see Standard Compliance, in the *RTI Connext Core Libraries User's Manual*).

However, request-reply and RPC may have ignored the environment variable and used the standard value. This issue did not affect communication using *Connext* products or connecting *Connext* with other vendors.

[RTI Issue ID REQREPLY-243]

5.4.15 XML Configuration

[Minor] Possible failure when loading an XML file containing a long comment

Loading an XML file may have failed if the file contained a long comment (longer than 2048 characters), if the comment was placed almost at the end of the XML file.

[RTI Issue ID CORE-15262]

5.4.16 Instances

[Major] Failure to register instances of mutable flat data types

`instance_to_keyhash` may have failed when *Connext* registered an instance. The error message may have looked something like the following:

```
ERROR PRESWriterHistoryDriver_registerInstance:!instanceToKeyHash
ERROR PRESPsService_registerInstance:!registerInstance
ERROR [0x0101E339,0xBC090EE0,0x31EC24EB:0x80000002{Entity=DW,Topic=TestTopic,
 ↪Type=flat_types::RecordWKey,Domain=229}|WRITE|ADD TO WRITER QUEUE]␣
 ↪PRESWriterHistoryDriver_addWrite:FAILED TO GET | Key hash
ERROR [0x0101E339,0xBC090EE0,0x31EC24EB:0x80000002{Entity=DW,Topic=TestTopic,
 ↪Type=flat_types::RecordWKey,Domain=229}|WRITE] PRESPsWriter_writeInternal:!
 ↪collator addWrite
write
```

The issue occurred if all of the following was true:

- FlatData was being used.
- The type was mutable.
- The *DataWriter's* `pool_buffer_max_size` property was set to a value less than the maximum serialized size of the key.

[RTI Issue ID CORE-15345]

5.4.17 Crashes

[Critical] Stack smashing error when serializing strings with RTI_CDR_SIZEOF_LONG_DOUBLE set to 16 in C++11 in release mode using GCC compiler

A stack smashing fault occurred when serializing strings if the `RTI_CDR_SIZEOF_LONG_DOUBLE` configuration was set to 16 in C++11 in release mode using the GCC compiler. Compiling the code reported a warning similar to:

```
include/ndds/hpp/rti/topic/cdr/InterpreterHelpers.hpp:165:31: note: the ABI␣
↪of passing union with ‘long double’ has changed in GCC 4.4
  165 |     static RTIXCdrMemberValue get_value_pointer(
      |                               ^~~~~~~~~~~~~~~~~
```

[RTI Issue ID CORE-14999]

[Critical] Possible crash on MIGInterpreter_parse while receiving data at the same time that entities are removed, on architectures with a weak memory model

On architectures that use a weak memory model, such as Arm or PowerPC, the application may have crashed when receiving data at the same time that the *DataReader* or a matching *DataWriter* was being removed. The crash happened on the function `MIGInterpreter_parse()` in some rare cases, due to a reordering of the memory storages. Now, the crash will not happen; the mentioned API can be called at the same time that a matching entity is removed.

[RTI Issue ID CORE-14923]

[Critical] Race condition in RTI Monitoring Library 2.0 caused a crash

A race condition in *RTI Monitoring Library 2.0* caused a segmentation fault in the application being monitored. The likelihood of the race condition occurring was extremely low.

[RTI Issue ID MONITOR-691]

[Critical] Segmentation fault if Application Kind contained more than 255 characters when calling RTI_DL_DistLogger_getInstance

If the Application Kind contained more than 255 characters, accessing the singleton instance of *RTI Distributed Logger* by the `RTI_DL_DistLogger_getInstance` method crashed with a segmentation fault.

[RTI Issue ID DISTLOG-244]

[Critical] Potential crash if creation of Publisher or Subscriber failed while database garbage collection process was running

RTI uses an internal in-memory "database" to store information about entities created locally as well as remote entities found during the discovery process. This database utilizes a background thread that periodically performs garbage collection on records related to deleted entities.

A potential crash could occur due to a race condition if the creation of a *Publisher* or *Subscriber* failed while the periodic garbage collection thread was active. Prior to the crash, an error message was logged in the `DDS_Subscriber_createI` or `DDS_Publisher_createI` functions.

The probability of encountering this race condition was low.

[RTI Issue ID CORE-15337]

[Critical] Possible crash when calling any write operation on architectures with a weak memory model

On architectures that use a weak memory model, such as Arm-based architectures, the application may have crashed when calling any write operation, `FooDataWriter_write()`, `FooDataWriter_write_w_timestamp` or `FooDataWriter_write_w_params` (or their C++ equivalents). The crash happened in some rare cases due to a reordering of the memory storages.

[RTI Issue ID CORE-15103]

[Critical] Stack overflow when parsing very long SQL filter expressions

If a SQL filter expression contained too many `OR`/`AND` expressions in a chain, parsing it may have caused a stack overflow and a program crash. For example:

```
a=1 OR a=2 OR a=3 OR a4 OR ... OR a=123456
```

[RTI Issue ID CORE-14985]

[Critical] Possible crash when creating DomainParticipants concurrently

Creating *DomainParticipants* concurrently could have caused a crash if, during the creation process, one of the threads accessed the typecode of the internal builtin types while another thread was initializing it.

[RTI Issue ID CORE-14764]

[Critical] Crash when writing data on a ContentFilteredTopic in combination with transient durability and delegated reliability

In scenarios where a ContentFilteredTopic was used, and the *DataWriter* and *DataReader* were using transient durability and delegated reliability (the `delegate_reliability` property is set to 1), a call to `write()` would crash if writer-side filtering was applied.

[RTI Issue ID CORE-15350]

[Major] Possible crash after memory allocation failures

A few cases were found where memory allocation failures were not properly handled. This could have led to null pointer dereferences if memory allocation failed.

[RTI Issue ID CORE-15024]

[Major] Using unbounded or very large types with query conditions in Python API could cause crash

If you created a query condition using the Python API and the serialized size of the samples that were evaluated against the query condition were large enough to cause a memory allocation failure, a crash could occur if the memory allocation failure was ignored and you continued to use the *DataReader* that generated the allocation failure.

[RTI Issue ID CORE-15029]

[Major] Segmentation fault if application ran out of memory while generating a log message

If the application ran out of memory while generating a log message, then the application would crash with a segmentation fault in the internal function `ADVLOGLogger_destroyMessageQueueElement`. This problem only affected releases 6.1.0 and above.

[RTI Issue ID CORE-15432]

5.4.18 Hangs

[Critical] Potential deadlock when using TCP Transport on Windows systems if WINDOWS_WAITFORMULTIPLEOBJECTS was selected as the socket monitoring API

A potential deadlock occurred when using TCP Transport on Windows systems if `WINDOWS_WAITFOR-MULTIPLEOBJECTS` was selected as the socket monitoring API.

[RTI Issue ID CORE-14983]

[Major] Potential deadlock when using Durable Writer History if there was a failure while creating the connection

A potential deadlock occurred when using Durable Writer History if there was a failure while creating the connection.

[RTI Issue ID CORE-15130]

[Major] Potential deadlock when using DynamicData in Debug libraries if TypePlugin initialization failed

A potential deadlock occurred when using DynamicData in the Debug libraries if the TypePlugin initialization failed.

[RTI Issue ID CORE-14986]

[Major] Potential issues using rti.asyncio.run or asyncio.run with dds.DataReader.take_async

There were two potential issues running the `take_async` methods:

- When the coroutine was run using `rti.asyncio.run`, attempting to explicitly close a `DomainParticipant` afterward may have caused an exception.
- When the coroutine was run using `asyncio.run`, a future call to `asyncio.run` for a new coroutine using the `take_async` methods may have deadlocked.

These two issues have been resolved. Furthermore, starting in this release, most applications may simply use `asyncio.run` instead of `rti.asyncio.run`.

[RTI Issue ID PY-181]

[Major] RPC failures when serializing the return value or out/inout parameters caused a timeout on client side

When an operation has a return value or out/inout parameters, serialization may fail under certain circumstances, such as if the return value is a bounded string and you return a string with a length that exceeds the bound. This failure produced a hang on the client side, ending with a `dds::core::TimeoutError` exception.

Now, the client will receive `dds::rpc::RemoteUnknownExceptionError`, letting the client know there's an error on the service side, rather than timing out. You can look at the service log to see the cause of the serialization error.

[RTI Issue ID REQREPLY-245]

5.4.19 Memory Leaks/Growth

[Critical] Possible unbounded memory growth on DataReader of keyed type after failing to match with a remote DataWriter

Consider the following scenario:

- You have a *DataReader* of a keyed type.
- The *DataReader* discovers a *DataWriter*, but the matching process fails somehow (e.g., due to a port collision that generates this error: `assertRemoteWriter:!create entryPort`).
- The *DataReader* somehow retries the matching process with the same *DataWriter* (e.g., due to receiving another discovery announcement for the *DataWriter* after the *DataWriter* changes its QoS).

The matching process would fail again and generate this error: `PRESCstReaderCollator_getRemoteWriterQueue:FAILED TO ASSERT | remote writer queue node in list`. Repeated occurrences of this error message were correlated with an unbounded memory growth on the *DataReader*.

This problem only affected *Connext* version 7.4.0, versions 7.1.0 to 7.3.0.2, versions 6.1.2.4 to 6.1.2.20, versions 6.0.1.33 to 6.0.1.38, and versions 5.3.1.43 to 5.3.1.44.

[RTI Issue ID CORE-15095]

[Critical] Memory leaks and errors when using DynamicDataReaders or FlatData DataReaders plus DDS-fragmentation and compression or encryption

When a DynamicDataReader or *DataReader* using a FlatData type received a fragmented sample that was either compressed or encrypted, the memory used to store the sample's serialized data was leaked and errors similar to the following would have been printed:

```
FATAL rCo79661##01Rcv [PARSE MESSAGE|0x01016350,0x5A66C0E7,
0x8DA940ED:0x80000004{Entity=DR,MessageKind=DATA_FRAG}|RECEIVE FROM
0x01018673,0xDB0C9361,0xB2B4181F:0x80000003] Mx02:/home/user/osapi.1.0/
srcC/memory/heap.c:1104:RTI0x2022004:inconsistent free/alloc: block id
0 being freed with "RTIOsapiHeap_allocateBufferAligned" and was allo-
cated with "RTIOsapiHeap_unknownFunction"
```

[RTI Issue ID CORE-15231]

[Critical] Unbounded memory growth when using DynamicDataReaders with multiple data representations and keyed types

There was an unbounded memory growth when a DynamicDataReader was using a keyed type and had the DataRepresentationQosPolicy set to `XCDR | XCDR2`. In order to run into the growth, the key of the data type had to contain a string or a sequence. The growth happened every time a sample was received from a *DataWriter* that was publishing data using XCDR DataRepresentation because the DynamicDataReader had to calculate the XCDR2 version of the keyhash. The memory used to calculate that keyhash was never released, leading to the unbounded growth.

[RTI Issue ID CORE-14437]

[Critical] Unbounded memory growth when using RECOVER instance state consistency and instances were removed from DataReader queue

If you set `ReliabilityQosPolicy::instance_state_consistency_kind=RECOVER`, each time an instance was removed from the *DataReader* queue some internal state about the remote writers of that instance was leaked.

Instances can be removed from the queue for the following reasons:

- An instance was disposed and `ReaderDataLifecycle::autopurge_disposed_instances_delay` was set to a finite value
- Or you set a finite `ResourceLimits::max_instances` value and configured `DataReaderResourceLimits::instance_replacement` to allow instance replacement when that limit was hit.

Note: If an instance was removed because there were no more writers for that instance, there was no leak because there was no remote writer information to leak about that instance.

[RTI Issue ID CORE-15091]

[Major] Memory leak when parsing invalid module in XML

A memory leak could be seen when trying to parse an invalid module in XML. For example:

```
<module name="MyModule1" autoid='hash' transferMode='invalid value'>
```

[RTI Issue ID CORE-15406]

[Major] Memory leak when using DynamicData with types containing optional members of string aliases or when cloning such types

Consider the following type definition:

```
typedef string EntityName;
struct Entity {
    @key uint64_t id;
    @optional EntityName name;
};
```

The following C code would have resulted in a 1-byte memory leak in previous releases:

```
dynData = DDS_DynamicData_new(
    Entity_get_typecode(),
    &DDS_DYNAMIC_DATA_PROPERTY_DEFAULT);
DDS_DynamicData_delete(dynData);
```

The problem occurred when the DynamicData object was associated with a type containing optional members of string aliases. The problem affected all languages.

The same problem also occurred when cloning a TypeCode for a type containing optional members of string aliases. For example, the following C code would have leaked 1 byte:

```
cloneTc = DDS_TypeCodeFactory_clone_tc(
        DDS_TypeCodeFactory_get_instance(),
        Entity_get_typecode(),
        &ex);
DDS_TypeCodeFactory_delete_tc(
        DDS_TypeCodeFactory_get_instance(),
        cloneTc,
        &ex);
```

[RTI Issue ID CORE-15539]

[Minor] DynamicDataTypeSupport objects not reclaimed by garbage collector

When a `DynamicDataTypeSupport` was registered with a *DomainParticipant*, the `DynamicDataTypeSupport` object was forever unreclaimable. This was the result of a leaked Java Native Interface (JNI) global reference.

[RTI Issue ID CORE-14359]

[Minor] Potential DynamicData sample leaked on DataReader deletion

When using DynamicData *DataReaders*, an internally-created sample might have been leaked if the DynamicDataTypeSupport was deleted before the *DataReader*.

[RTI Issue ID CORE-14324]

5.4.20 Data Corruption

[Critical] Potential corruption in samples with sequences of strings when using Traditional C++ API and std::string

When using the Traditional C++ API and generating code using `-useStdString`, received samples with fields of sequences of strings (`sequence<string>`) may have contained corrupted elements. Specifically, the elements (`std::string`) could have more characters than expected, due to a desynchronization between the C++ string and its underlying buffer, such as `x.length() != std::strlen(x.c_str())`.

[RTI Issue ID CORE-15229]

[Critical] Rare race condition may have led to undefined behavior

The fix for RTI Issue ID CORE-14893 in 7.4.0 (see *Other*) was incomplete; there were still some conditions that could have led to the race condition.

[RTI Issue ID CORE-15199]

[Major] Possible data corruption when using ISerializer<T> to serialize a sample

When using an `ISerializer<T>` object with `Xcdr1` data representation, operations such as serializing or retrieving the serialized sample's size may have produced incorrect results.

[RTI Issue ID CORE-14933]

5.4.21 Entities

[Major] Error when setting QoS on Publisher or Subscriber without endpoints

When changing the PartitionQosPolicy of *Publishers* or *Subscribers* without endpoints (*DataWriters* or *DataReaders*), the operation could have unexpectedly failed and logged this error:

```
FAILURE | Notify of group or partition change
```

The likelihood of running into this issue increased with the number of *Publishers* or *Subscribers* created by your application.

[RTI Issue ID CORE-15322]

5.4.22 Interoperability

[Critical] XCDR serialization of a mutable union with a discriminator value that does not select a union member was not compliant with OMG specification

A union's discriminator value may not select any member in the union. When this was the case, and when the union was mutable, the XCDR serialization of the union was not compliant with the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3, because it was missing the 4-byte sentinel value `02 7f 00 00` at the end. Now, you may configure whether or not the sentinel value is present by using a new "sentinel in empty union" bit in the XTypes compliance mask. Here is an example using the Modern C++ API:

```
using namespace rti::config;

compliance::set_xtypes_mask(compliance::get_xtypes_mask() |␣
↪compliance::XTypesMask::sentinel_in_empty_union());
```

To maintain compliance with the XTypes specification, this bit is set by default. To maintain backward compatibility with previous versions of *Connext*, you must explicitly unset this bit. For instructions, see Extensible Types Compliance Mask in the *RTI Connext Core Libraries Extensible Types Guide*.

This problem only affected *Connext* versions 6.0.0 and above. This problem affected Dynamic Data in all language APIs and generated code in all language APIs except for Java. *DataReaders* from versions prior to 6.0.0 may encounter deserialization failures when receiving samples from a *DataWriter* in version 6.0.0 or later if mutable unions are used in the Topic type and if the *DataWriter* is not using a version that has the fix for this issue.

[RTI Issue ID CORE-14997]

[Critical] RTPS messages containing vendor-specific submessages could have been dropped

When *Connext* received RTPS messages containing vendor-specific submessages, *Connext* could have dropped the whole RTPS message mid-processing. This happened when non-*Connext* applications used a vendor-specific submessage whose ID clashed with an ID used by *Connext*.

Now, *Connext* will drop RTPS submessages from non-*Connext* applications whose Submessage Id is within the vendor-specific range (`[0x80, 0xFF]`). This ensures that the rest of the RTPS message is processed, which may still contain RTPS standard submessages that *Connext* can interpret.

[RTI Issue ID CORE-15265]

[Critical] Multiple RTPS messages received in a single datagram caused issues

Previously, multiple RTPS messages received in a single datagram could cause issues ranging from incorrect attribution to processing errors, RTPS-protection decoding errors, and no communication. Multiple messages in a single buffer are now accepted and successfully parsed, provided they contain an RTPS Header Extension with the RTPS message length correctly populated.

[RTI Issue ID CORE-14701]

5.4.23 Vulnerabilities

The following vulnerabilities are fixed in this release.

See also RTI Connext Security Bulletins and Advisories for a complete list of vulnerabilities in RTI releases that have been published through the CVE® Program. That list may be more up-to-date.

[Critical] Potential stack buffer overflow in Connext applications when parsing an XML type

The stack may have been corrupted when parsing an XML type.

User Impact without Security

This vulnerability could have caused the following on any application using XML types:

- Stack corruption leading to data corruption or crash.
- Unbounded memory growth.
- Exploitable through malicious RTPS messages.
- Exploitable through a compromised local file system containing a malicious XML file.
- CVSS v3.1 Base Score: 9.1 CRITICAL
- CVSS v3.1 Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H
- CVSS v4.0 Base Score: 8.3 HIGH
- CVSS v4.0 Vector: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:N/VI:H/VA:H/SC:N/SI:N/SA:N

User Impact with Security

This vulnerability could have caused the following on any application using XML types:

- Stack corruption leading to data corruption or crash.
- Unbounded memory growth.
- Exploitable through a compromised local file system containing a malicious XML file.
- CVSS v3.1 Base Score: 7.1 HIGH
- CVSS v3.1 Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H
- CVSS v4.0 Base Score: 6.9 MEDIUM
- CVSS v4.0 Vector: CVSS:4.0/AV:L/AC:L/AT:N/PR:L/UI:N/VC:N/VI:H/VA:H/SC:N/SI:N/SA:N

[RTI Issue ID CORE-14870]

[Critical] Potential stack buffer write overflow in license-managed Core Libraries when setting RTI_LICENSE_FILE environment variable

The stack may have been corrupted while loading the `RTI_LICENSE_FILE` environment variable.

User Impact without Security

This vulnerability could have caused the following on any application loading the license information through the `RTI_LICENSE_FILE` environment variable:

- Stack corruption leading to data corruption or crash.
- Exploitable through a compromised local file system containing a malicious license file referenced by the `RTI_LICENSE_FILE` environment variable.
- CVSS 3.1 Base Score: 7.1 HIGH
- CVSS 3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H
- CVSS 4.0 Base Score: 6.9 MEDIUM
- CVSS 4.0 Vector: AV:L/AC:L/AT:N/PR:L/UI:N/VC:N/VI:H/VA:H/SC:N/SI:N/SA:N

User Impact with Security

This vulnerability could have caused the following on any application loading the license information through the `RTI_LICENSE_FILE` environment variable:

- Stack corruption leading to data corruption or crash.
- Exploitable through a compromised local file system containing a malicious license file referenced by the `RTI_LICENSE_FILE` environment variable.
- CVSS 3.1 Base Score: 7.1 HIGH
- CVSS 3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H
- CVSS 4.0 Base Score: 6.9 MEDIUM
- CVSS 4.0 Vector: AV:L/AC:L/AT:N/PR:L/UI:N/VC:N/VI:H/VA:H/SC:N/SI:N/SA:N

[RTI Issue ID CORE-15310]

[Critical] Buffer write overflow while parsing malicious license file

An out-of-bounds write on the heap could occur while parsing a malicious license file.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Heap buffer overflow while parsing a malicious license file.
- Exploitable by overwriting the license file on the file system with a malicious license file.
- Potential impact on integrity and availability of *Connext* application.
- CVSS Base Score: 7.1 HIGH

- CVSS v3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

User Impact with Security

Same impact as "User Impact without Security" above.

[RTI Issue ID CORE-15145]

[Critical] Potential integer overflow in Connext applications on 32-bit systems when parsing XML files with very large number of default attributes or levels of nesting

The Core Libraries XML parser had a third-party dependency on Expat version 2.6.2, which is known to be affected by a number of publicly disclosed vulnerabilities. These vulnerabilities have been fixed by upgrading Expat to the latest stable version, 2.6.3. See the "What's New" section in this document for more details.

The impact on *Connext* applications of using the previous version varied depending on your *Connext* application configuration:

User Impact without Security

- Exploitable through a compromised local file system containing malicious XML/DTD files.
- Remotely exploitable through malicious RTPS messages.
- If exploited, impact ranged from denial of service to potentially arbitrary code execution.
- CVSS v3.1 Score: 9.8 CRITICAL
- CVSS v3.1 Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

User Impact with Security

- Exploitable through a compromised local file system containing malicious XML/DTD files.
- If exploited, impact ranged from denial of service to potentially arbitrary code execution.
- CVSS v3.1 Score: 8.4 HIGH
- CVSS v3.1 Vector: AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

[RTI Issue ID CORE-15121]

5.4.24 Other

[Critical] Data races on architectures with weak memory models

On systems with weak memory architectures, such as Arm® and PowerPC®, data races could have led to undefined behavior or crashes in various *Connext* components.

[RTI Issue ID CORE-15002]

[Major] Printing an IDL union with an enum discriminator could have produced an incorrect output

When printing an IDL union with an enum discriminator, the resulting case labels may have been incomplete if the enum types were defined in a separate module.

[RTI Issue ID CORE-14654]

[Major] Incorrect CPU assignment when using THREAD_SETTINGS_CPU_RR_ROTATION in receiver_pool

There was an incorrect CPU assignment for the receive threads when using the THREAD_SETTINGS_CPU_RR_ROTATION setting in the receiver_pool. Specifically, all the receive threads were assigned to the first CPU listed in the cpu_list.

[RTI Issue ID CORE-13211]

[Trivial] Possible data race when creating DomainParticipants

During the creation of a *DomainParticipant*, it was possible to run into a data race due to the use of the `localtime` or `gmtime` API by other threads in *Connext*. There was no functional impact of this data race; however, it has been fixed by using the safe versions of those APIs available on the different operating systems.

[RTI Issue ID CORE-14877]

[Trivial] Printing a type with an annotation using an enum produced an incorrect output

When printing a type with an annotation using an enum, the resulting annotations contained the ordinal instead of the enum type name.

[RTI Issue ID CORE-14729]

5.5 What's New in 7.4.0

This section describes what's new in the Core Libraries compared to release 7.3.0.

RTI® Connext® 7.4.0 is an early access release. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connext* suite, see those products' release notes on the RTI Community Portal or in your installation. Or start with the RTI Connext What's New for a launchpad to all products.

Note: For backward compatibility information between 7.4.0 and previous releases, see the Migration Guide on the RTI Community Portal.

5.5.1 Set up reliable communications for fragmented data more easily and with better performance, by removing asynchronous publishing requirement

In earlier releases of *Connext*, if your data exceeded the transport's configured maximum message size (`message_size_max`) and required fragmentation, and your *DataWriter* was communicating with a reliable *DataReader*, you had to manually enable asynchronous publishing on the *DataWriter*. If not enabled, *Connext* would display a warning, and communication would not occur:

> *"WARNING Max serialized size of type 'your_type' (your_type_max_serialized_size) exceeds the transport's message size max (your_message_size_max_configuration). Consider using asynchronous publishing or increasing message size max otherwise samples with a serialized size larger than 'your_message_size_max_configuration' will not be able to be sent."*

From *Connext* release 7.4.0 onward, sending fragmented data reliably with synchronous *DataWriters* has become possible. This functionality, previously available only for best-effort communications, has now been extended to support reliable endpoints as well.

This enhancement not only simplifies your workflow by removing the need to pre-configure asynchronous publishing based on data fragmentation requirements, but also leverages the generally lower latency of synchronous *DataWriters*.

As part of this work, the following RtpsReliableWriterProtocol_t QoS policy settings, which previously applied only to DDS samples, now apply to both DDS samples and DDS fragments:

- `min_nack_response_delay` and `max_nack_response_delay` set the boundaries of a time interval within which missing samples must be repaired. This interval allows for any additional `NACKs` (for missing samples) and `NACK_FRAGs` (for missing data fragments) that are received during that time to be coalesced and responded to all at once instead of individually.
- `nack_suppression_duration` specifies a duration within which subsequent duplicate `NACKs` or `NACK_FRAGs` are disregarded. If a duplicate `NACK` or `NACK_FRAG` is received within this duration, it will not trigger another repair process.
- `max_bytes_per_nack_response` allows for basic flow control of repair data by limiting the number of bytes that will be sent in response to a `NACK` or `NACK_FRAG`. A *DataWriter* will repair the miss-

ing samples until this limit is reached, always repairing at least one sample. The samples that were not repaired will be repaired in the next received `NACK` or `NACK_FRAG`.

The enhancement to these settings means you can now shape the traffic of data fragment repairs without relying on a flow controller (asynchronous publishing) and still benefit from the advantages of using synchronous publishing.

For background information on these topics, see Sending Large Data, ASYNCHRONOUS_PUBLISHER QosPolicy (DDS Extension), and Data Fragmentation in the *RTI Connext Core Libraries User's Manual*.

5.5.2 Modern C++ RPC API now fully productized and supported

This release promotes the RPC C++ API from experimental to production-ready. It includes several improvements to the RPC and Request-Reply Modern C++ APIs that align the API with the future update of the OMG DDS-RPC standard and provide several new features, most notably, an enhanced service discovery protocol.

The enhanced service discovery protocol allows a Client (or a Requester) to reliably discover a Service (or a Replier) before making a call:

- Clients and Requesters can call a new method, `wait_for_service()`, to ensure that a service has been discovered.
- Clients and Requesters will fail immediately when making a request if no service has been discovered. This behavior can be disabled on the Requester with the new `RequesterParams` option, `require_matching_service_on_send_request`, for interoperability with request-reply APIs in other languages and previous versions of the C++ API.
- A Service or a Replier will only pass requests to the application layer when it can ensure that the reply can be delivered. This solves a previous race condition that could cause some initial replies for a Client or Requester to be lost.

Other changes include:

- Changes the translation of service interfaces to topic-types to align with a future update of the OMG RPC-DDS standard.
- Allows for the mutability of RPC operations by supporting the @final and @mutable annotations on a per-operation basis.

  For example, the following interfaces are compatible. A Foo client can call `my_method` on a Bar service, and the other way around (as long as they use the same service name):

```
@service("DDS")
interface Foo {
      @mutable
      long my_methodop(in short s, out double d);
};

@service("DDS")
interface Bar {
     @mutable
     void my_method(in short s, out double d, in long l);
};
```

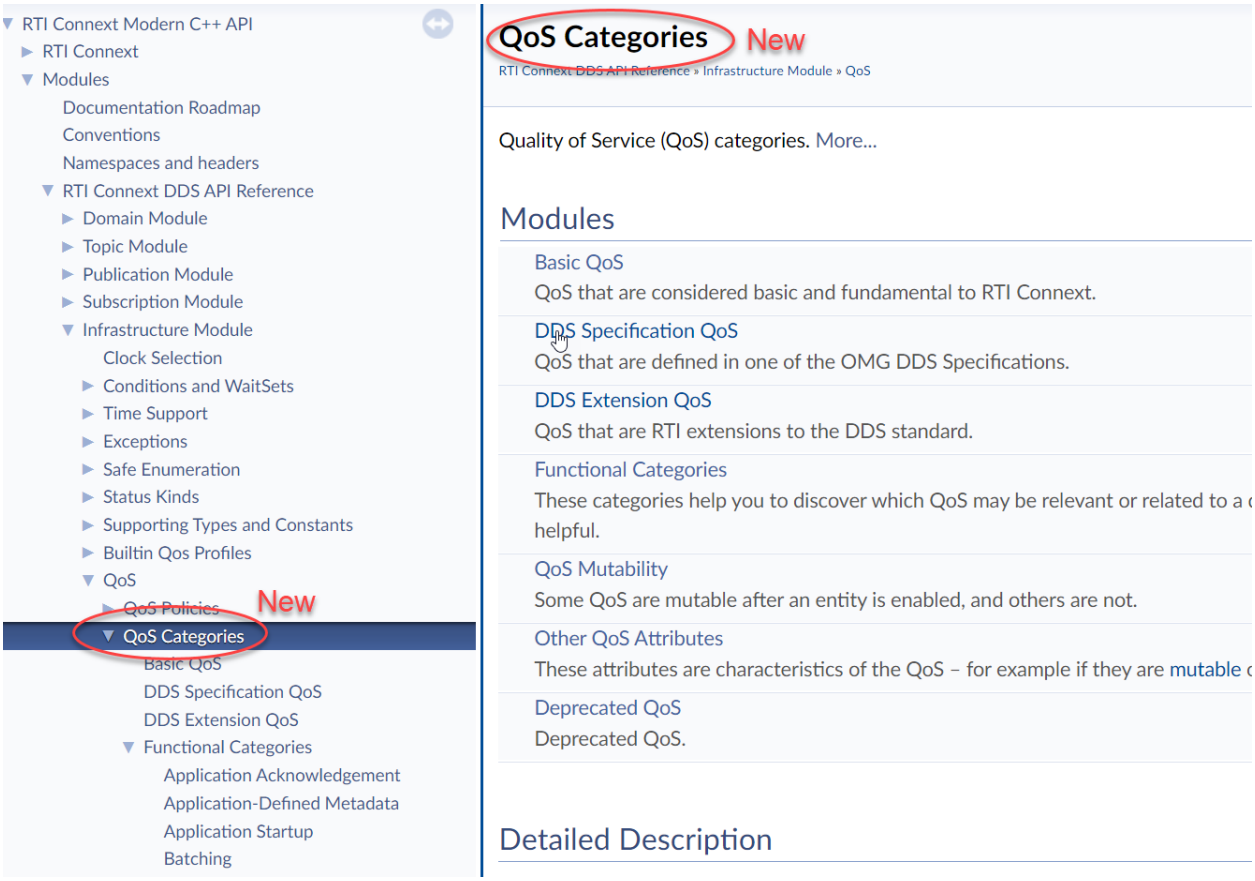
- Allows the propagation of built-in exceptions: `dds::rpc::RemoteUnsupportedOperationError`, `dds::rpc::RemoteInvalidArgumentError`, `dds::rpc::RemoteOutOfResourcesError`, `dds::rpc::RemoteUnknownOperationError`, `dds::rpc::RemoteUnknownExceptionError`, in addition to the already-supported user-defined exceptions. These built-in exceptions can be explicitly thrown by the service implementation. Additionally, `dds::rpc::RemoteUnknownOperationError` and `dds::rpc::RemoteUnknownExceptionError` are reported when the client calls an unknown operation in the service or the service implementation throws an unknown exception, respectively.
- Allows setting wire compatibility with `rti::config::compliance::RpcMask` so that the PID_RELATED_SAMPLE_IDENTITY RTPS parameter is set to its standard value. By default, it continues to be set to a non-standard value for backward compatibility with previous *Connext* versions.

Note: The RPC Python API continues to be experimental in release 7.4.0, but will be production-ready in 7.5.0. (The Request-Reply Python API *is* production-ready.)

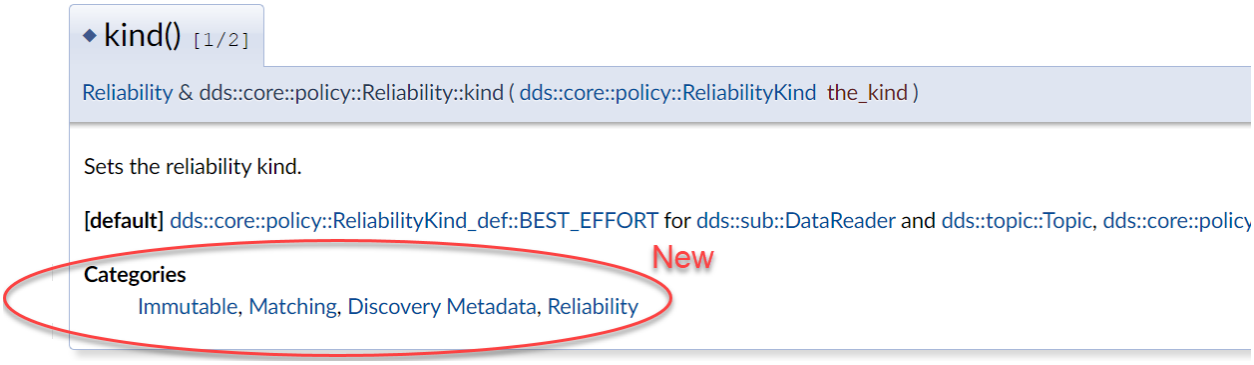
See Request-Reply Exchanges and Remote Procedure Calls (RPC) in the *RTI Connext Core Libraries User's Manual* for information on these APIs.

5.5.3 Discover relevant QoS more easily through categorizations added to API Reference HTML documentation

Connext Qualities of Service (QoS) policies and fields have been grouped into categories.



The categories are also listed with each QoS field's documentation in the API HTML Documentation.



These categories can help you discover which QoS may be relevant or related to a desired functionality or feature. For example, if you are interested in Sending Large Data, then looking at any QoS in that category will be helpful. The different functional categories are listed in the API HTML Documentation under **Modules > RTI Connext DDS API Reference > Infrastructure > QoS > QoS Categories > Functional Categories**. For example, see Functional Categories in the Modern C++ API Reference.

Some categories are not necessarily related to achieving a specific functionality, but are characteristics of the QoS – for example, if the QoS are mutable or not, or if they apply per-instance. These categories are listed under the following headings in the API Reference:

- **Modules > RTI Connext DDS API Reference > Infrastructure > QoS > QoS Categories >** QoS Mutability
- **Modules > RTI Connext DDS API Reference > Infrastructure > QoS > QoS Categories >** Other QoS Attributes

5.5.4 Easier QoS configuration for Request-Reply and RPC with new built-in QoS profile

This release includes a new built-in profile, `BuiltinQosLib::Pattern.RPC` that can be used to obtain the default values used to create the *DataReaders* and *DataWriters* within a Requester, Replier, RPC Client, or RPC Service.

You can use this profile as the base profile in your XML configuration. Find an example that uses the built-in profile in the GitHub Examples repository: https://github.com/rticommunity/rticonnextdds-examples/blob/master/examples/connext_dds/remote_procedure_call/c%2B%2B11/USER_QOS_PROFILES.xml.

Our you can use the profile in code. For example:

In C++:

```
auto qos_provider = dds::core::QosProvider::Default();
auto rpc_reader_qos = qos_provider.datareader_qos(
    rti::core::builtin_profiles::qos_lib::pattern_rpc());
auto rpc_writer_qos = qos_provider.datawriter_qos(
    rti::core::builtin_profiles::qos_lib::pattern_rpc());

// modify rpc_reader_qos/rpc_writer_qos...
```

```
rti::request::Requester<Foo, Bar> requester(
            rti::request::RequesterParams(participant)
                    .service_name("my service")
                    .datareader_qos(rpc_reader_qos)
                    .datawriter_qos(rpc_writer_qos);
```

In Python:

```
import rti.rpc as rpc
import rti.connextdds as dds

rpc_reader_qos = dds.QosProvider.default.datareader_qos_from_profile(
    dds.BuiltinProfiles.pattern_rpc
)
rpc_writer_qos = dds.QosProvider.default.datawriter_qos_from_profile(
    dds.BuiltinProfiles.pattern_rpc
)

# modify rpc_reader_qos/rpc_writer_qos...

requester = rpc.Requester(
    Foo,
    Bar,
    participant,
    "my service",
    datareader_qos=rpc_reader_qos,
    datawriter_qos=rpc_writer_qos,
)
```

5.5.5 Write Python applications faster with reactive subscriptions

This release adds a new feature to the Connext Python API that allows subscribing to a topic by simply decorating a function that receives the updates for that topic.

For example, the following is a full application that subscribes to two topics, "Sensor Temperature" and "Alerts," and prints the data received for each topic:

```
import rti.asyncio
from my_types import Temperature, Alert

app = rti.asyncio.Application()

@app.subscribe("Sensor Temperature")
async def on_sensor_temperature(data: Temperature):
    print("Received Sensor Temperature:", data)

@app.subscribe("Alerts")
async def on_alert(data: Alert):
    print("Received Alert:", data)
```

```
if __name__ == "__main__":
    rti.asyncio.run(app.run(domain_id=0))
```

`rti.asyncio.Application` is a new class that creates a `Topic` and a `DataReader` for any `async` function decorated with its `subscribe` decorator, and passes all data updates for that topic to the function. The topic type is inferred from the function's argument type annotation.

This feature allows quickly writing Python applications that subscribe to one or more topics.

For more information, see the Python API Documentation.

5.5.6 New Ping built-in type helps debug and prototype with the Python API

This release adds a new built-in type to the Python API, `rti.types.builtin.PingType`. This is the type that the *rtiddsping* command-line utility publishes and subscribes to and consists of a single 32-bit integer. This type helps you quickly interact with *rtiddsping* from a Python application to debug or prototype without having to define the type yourself. For example, the following Python script prints the data published by *rtiddsping*:

```
from rti.types.builtin import PingType
import rti.asyncio

app = rti.asyncio.Application()

@app.subscribe("PingTopic")
async def on_ping(ping: PingType):
    print("Received ping:", ping.number)

rti.asyncio.run(app.run(domain_id=0))
```

(This snippet uses the `rti.asyncio.Application` utility for simplicity, which is also new in this release.)

And you can publish that topic by just running *rtiddsping* with no arguments:

```
$ rtiddsping
```

For more about the built-in types in the Python API, see rti.types.builtin in the Python API reference.

5.5.7 DDS Spy now prints full timestamps (with date and fractions of a second) and epoch timestamps

In previous releases, *rtiddsspy* displayed timestamps in the `hh:mm:ss` format, which didn't include the date or fractions of a second.

This release adds a new option, `-timeFormat SHORT | FULL | EPOCH`:

- `-timeFormat SHORT` is the default and prints in `hh:mm:ss` format.
- `-timeFormat FULL` adds the date and fractions of a second: `yy:mm:dd hh:mm:ss.s`.

- `-timeFormat EPOCH` displays the time as the number of seconds since the Unix epoch.

For example:

```
$ rtiddsspy -printSample COMPACT -timeFormat FULL

2024-05-24 01:35:35.154618 New writer        from 10.0.0.225      : topic=
↪"PingTopic" type="PingType"
2024-05-24 01:35:35.154836 New data          from 10.0.0.225      : topic=
↪"PingTopic" type="PingType" sample={"number":0}
2024-05-24 01:35:36.159999 New data          from 10.0.0.225      : topic=
↪"PingTopic" type="PingType" sample={"number":1}
```

See the DDS Spy User's Manual for more information.

5.5.8 New DynamicData APIs get and set a union's discriminator value

Two new DynamicData APIs have been added that allow getting and setting a union's discriminator value: `DynamicData::get_discriminator()` and `DynamicData::set_discriminator()`.

Previously, passing index 0 to the `DynamicData::get_member_info_by_index()` API returned the discriminator value for a union in the **DynamicDataMemberInfo::member_id** field. This behavior has been deprecated and the `DynamicData::get_discriminator()` API should be used instead.

The **DynamicData::set_<type>()** APIs used to be the only way to set the discriminator value, by providing the desired discriminator value as the **member_id**. Now, the discriminator value can also be set using the `DynamicData::set_discriminator()` API.

See Accessing the Discriminator Value in a Union in the *Core Libraries User's Manual* and the API documentation for more information about these APIs and examples of how to use them.

5.5.9 Deprecations and Removals

This section describes items that are deprecated or removed in 7.4.0, compared to 7.3.0.

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

Any deprecations or removals noted in RTI's documentation 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. RTI's current standard terms and support and maintenance policies are available at https://www.rti.com/terms.

Simplify QoS configuration by deprecating unnecessary or rarely used QoS settings and policies

As part of the effort to simplify Quality of Service (QoS) configuration in *Connext*, this release has deprecated a number of QoS settings and policies because their use cases are narrow or they are seldom used. Although still available, these QoS may be removed in a future release, handled by new functionality, or hidden. These QoS are marked as deprecated in the API Reference and *RTI Connext Core Libraries User's Manual*.

To see the full list of deprecated QoS settings, see the API Reference HTML documentation under **Modules > RTI Connext DDS API Reference > Infrastructure > QoS > QoS Categories > Deprecated QoS**. For example, see Deprecated QoS in the Modern C++ API Reference.

dds.sample_assignability.accept_unknown_enum_value and dds.sample_assignability.accept_unknown_union_discriminator properties deprecated

As described in *OMG Specification Compliance*, the properties `dds.sample_assignability.accept_unknown_enum_value` and `dds.sample_assignability.accept_unknown_union_discriminator` have been deprecated and replaced with Extensible Types compliance mask bits. See Extensible Types Compliance Mask in the *Core Libraries Extensible Types Guide* for more information.

If you are currently using the properties, see the 7.4.0 Migration Guide on the RTI Community Portal for information on configuring their behavior using the Extensible Types compliance mask bits.

dds.participant.use_45d_compatible_alignment_enforcement property deprecated

The `dds.participant.use_45d_compatible_alignment_enforcement` property is deprecated in this release. It is still functional in this release and can be used when interoperating with *Connext* 4.5d or earlier. It should not be used when your system does not include any legacy (earlier than 4.5d) *Connext*-based applications.

EXCLUSIVE_AREA QoS policy removed

The EXCLUSIVE_AREA QoS policy was deprecated in release 6.1.1. In release 7.3.0, it was announced to no longer be supported, and its documentation was removed. In this release, it is fully removed (all user-accessible code associated with the QoS has been removed). This removal only affects the EXCLUSIVE_AREA QoS policy (the ability to set **use_shared_exclusive_area**).

Deprecated behavior in DynamicData API

Previously, passing index 0 to the `DynamicData::get_member_info_by_index()` API returned the discriminator value for a union in the **DynamicDataMemberInfo::member_id** field. This behavior has been deprecated and the `DynamicData::get_discriminator()` API should be used instead.

See *New DynamicData APIs get and set a union's discriminator value* above for more information.

DDS-prefixed enumerations removed from XSD files

As described in *[Trivial] DDS-prefixed enumerations caused confusion in XSD files*, DDS-prefixed enumerations such as DDS_KEEP_LAST_HISTORY_QOS and DDS_KEEP_ALL_HISTORY_QOS have been removed, although they remain valid within the *Connext* XML parser. Anyone leveraging third-party tools for XML file validation might encounter warnings or errors if the removed DDS-prefixed enumerations are still in use. Find more information in the *Migration Guide* here.

5.6 What's Fixed in 7.4.0

This section describes bugs fixed in *Connext* 7.4.0. These are fixes since 7.3.0.

Connext 7.4.0 is an early access release. See the Connext Versions and Lifecycle page for more information on RTI's software release model.

[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.

5.6.1 Discovery

[Critical] DNS Tracking not supported with participants using SPDP2

DNS tracking was not supported for *DomainParticipants* using SPDP2. The *DomainParticipant* continued sending bootstrap announcements to the IP address of newly resolved hostnames, even if existing participants were using the IP address, resulting in excess traffic.

The *DomainParticipant* may have also failed to resume sending bootstrap announcements to the IP address that a hostname was resolved to once existing remote participants using that IP address lost liveliness. This caused a failure to discover new remote participants at those IP addresses.

DNS tracking is now supported when using SPDP2.

[RTI Issue ID CORE-14113]

5.6.2 Serialization and Deserialization

[Major] Samples containing sequences using discontiguous buffers failed to be serialized

Sending a sample containing a sequence that used a discontiguous buffer as the underlying storage for its elements failed with an error similar to this:

```
RTIXCdrLog_logWithParams:!precondition: sample == ((void *)0)
RTIXCdrInterpreter_fullSerializeSample:MyContainer:m_instance serialization␣
↪error
```

[RTI Issue ID CORE-10485]

[Major] Calculated maximum size incorrect for unions with nested unions whose members were empty structures

Consider an IDL with a union that has a nested union, and all the nested union members are empty structures, like this:

```
struct empty {};

union unionWithEmptyStruct switch (int32) {
    case 0:
      empty e;
};

@appendable
union unionWithNestedUnionsWithEmptyStruct switch (int32) {
    case 2:
      unionWithEmptyStruct memberWithEmptyStructure;
};
```

The method to calculate the maximum serialize sample size returned RTI_CDR_MAX_SERIALIZED_SIZE. Since the maximum serialize size defines the space needed for serialization/deserialization buffers, returning RTI_CDR_MAX_SERIALIZED_SIZE could cause an excessive use of memory.

The calculation of the maximum serialize sample is now correct.

[RTI Issue ID CORE-14760]

5.6.3 Usability

[Major] MultiChannel did not work for synchronous DataWriters with large data in best-effort communications

A best-effort *DataReader* was not able to receive large data samples (samples whose serialized size is bigger than the transport `message_size_max`) from a *DataWriter* using MultiChannel and synchronous publish mode.

In that specific case, the original identity of the samples was not preserved. Therefore, the *DataReader* rejected the samples as duplicated.

Note that this issue was incorrectly marked as fixed in 6.1.0; in fact, MultiChannel was only fixed for asynchronous *DataWriters*. Now it is fixed for any combination of reliability and publish mode.

[RTI Issue ID CORE-14668]

5.6.4 Transports

[Major] Potential high CPU usage if network stack was deleted

The deletion of the network stack while a *Connext* application using UDP transports was running could lead to high CPU usage. The high CPU usage has been mitigated, but other problems may still occur. You should restart your *Connext* application once the network stack is running again.

[RTI Issue ID CORE-14879]

[Minor] Wrong configuration of transport_priority QoS on TCP Transport

If the value of the property `dds.transport.TCPv4.tcp1.transport_priority_mapping_low` was higher than the value of `dds.transport.TCPv4.tcp1.transport_priority_mapping_high`, the `transport_priority` QoS setting for the TCP Transport was configured incorrectly. The values of these properties are now checked during property validation of the TCP Transport. If they are configured incorrectly, you will receive an error.

[RTI Issue ID COREPLG-737]

5.6.5 Reliability Protocol and Wire Representation

[Critical] Repair sample was not sent if the sample was smaller than either transport's message_size_max

This issue was fixed in previous releases but not documented at those times.

Consider the following scenario:

- The *DomainParticipant* is using two transports.
- The transports have different values for `message_size_max`.
- The *DataWriter's* `publish_mode.kind` is `SYNCHRONOUS`.
- The *DataWriter* writes a sample whose serialized size is between the two values of `message_size_max`.
- The *DataReader's* `reliability.kind` is `RELIABLE`.

In release 5.0.0, this sample was successfully sent as live data using the transport with the larger `message_size_max`. But if this sample ever had to be resent as repair data, the resend attempt would fail with the following error message:

```
!write resend. Reliable large data requires asynchronous writer
```

The problem was the inconsistency between the behavior of live data and repair data. This problem affected release 5.0.0. A fix was made in 5.3.0 (undocumented) so both the live data and repair data were successfully sent using the transport with the larger `message_size_max`.

Further changes were made in 5.3.1.20 and 6.0.1, as part of RTI Issue ID CORE-9287. Starting in those releases, both live data and repair data in the above scenario failed to be sent because one of the `message_size_max` values was too small for the sample. The send attempt failed with the following error message:

```
COMMENDFacade_canSampleBeSent:NOT SUPPORTED | Reliable fragmented data
requires asynchronous writer.
```

Starting in release 7.4.0, because of the new feature *Set up reliable communications for fragmented data more easily and with better performance, by removing asynchronous publishing requirement*, the sample is successfully sent as live and repair data, regardless of the value of the `message_size_max` of the installed transports.

[RTI Issue ID CORE-9297]

[Critical] Late-joiner DataReader may have stopped receiving samples from DataWriters using a finite durability.writer_depth

A late-joining *DataReader* may have stopped receiving samples from a *DataWriter* configured with a finite `durability.writer_depth`. This issue might have occurred under the following conditions:

- The *DataReader* was using multicast
- The *DataWriter* was configured to use an asynchronous publisher

When the issue occurred, you would have observed an infinite exchange of Heartbeat messages, followed by NACK messages, on the wire without the *DataWriter* re-sending the samples mentioned in the NACK messages.

[RTI Issue ID CORE-14930]

[Major] inactivate_nonprogressing_readers did not always inactivate non-progressing readers

The RtpsReliableWriterProtocol_t::inactivate_nonprogressing_readers field did not always work as expected. It was possible for a non-progressing *DataReader* to incorrectly stay activated despite setting this field to `true`. Most of the time, this problem took effect when the *DataReader* was not progressing beyond the initial periodic NACKs (see RtpsReliableReaderProtocol_t::nack_period).

[RTI Issue ID CORE-14425]

[Major] Reliable large data performance issues due to redundant fragment repairs

In this scenario, while publishing large data reliably, a *DataWriter* was sending a sample and the *DataReader* had received some fragments from that sample. If the *DataReader* received a heartbeat message, the *DataReader* may have requested all the fragments missing from that sample (by sending a NACK_FRAG message), including the ones that had not been sent yet.

When the *DataWriter* received the NACK_FRAG message, this triggered the *DataWriter* to send redundant repair messages for those fragments that had not been sent. This behavior affected the *DataWriter's* performance. Now the *DataReader* only requests the fragments that the *DataWriter* has sent and are missing.

[RTI Issue ID CORE-14599]

[Major] DataWriters with finite durability.writer_depth may have sent repairs over unicast instead of multicast

A *DataWriter* configured with finite `durability.writer_depth` is intended to use unicast to send repairs to a late-joining *DataReader* configured with a multicast address, then switch to multicast once the *DataReader* has caught up. This switch from unicast to multicast did not occur as expected. As a result, late-joining *DataReaders* may have continued receiving repair data via unicast instead of multicast even after catching up, potentially affecting network performance.

The issue only affected *DataWriters* using batching or multichannel features.

[RTI Issue ID CORE-14823]

5.6.6 Bandwidth Sensitivity

[Minor] Excessive bandwidth consumption with SPDP2 for peers that were added multiple times

DomainParticipants configured to use Simple Participant Discovery Protocol 2.0 (SPDP2) did not stop sending bootstrap messages to a locator if the peer associated with the locator was added multiple times to the *DomainParticipant*. This could occur, for example, when the `DDS_DomainParticipant_add_peer()` API was invoked multiple times with the same peer.

[RTI Issue ID CORE-14920]

5.6.7 Debuggability

[Minor] Receiving large sequence numbers caused DataReader to incorrectly report on_sample_lost() with a negative total_count

When receiving any message that triggered a *DataReader* to report lost samples (for example, when the *DataReader* has BEST_EFFORT reliability and receives a sample whose sequence number is much larger than the previously received sample), the *DataReader* would have reported a SampleLostStatus with a negative value for the `total_count` if this message caused the `total_count` to exceed `0x7fffffff`. This

problem has been fixed by imposing a maximum value of `0x7fffffff` for the `total_count`. The same problem and fix apply to `total_count_change` as well.

Note that RTI Issue ID CORE-8921 (which was fixed in release 6.0.0) described a more specific scenario in which the `total_count` could have become negative. In this release, the problem has been fixed more generally.

[RTI Issue ID CORE-14673]

5.6.8 Content Filters and Query Conditions

[Critical] Failure filtering samples on a DataReader when using DynamicData and Zero Copy

A DynamicData *DataReader* using a ContentFilteredTopic or QueryConditions and communicating with a *DataWriter* using *Zero Copy transfer over shared memory* may have failed to filter the received samples when the *DataReader's* DynamicData TypeSupport was configured to skip sample deserialization by setting the option **DynamicDataTypeSerializationProperty.skip_deserialization** to true.

When this problem occurred, the *DataReader* generated the following error messages:

```
ERROR [0x0101BE35,0x47B28D9B,0x87A311F6:0x80000004{Entity=DR,MessageKind=DATA}
 ↪|RECEIVE FROM 0x0101D676,0xE92E1D8E,0x91728D11:0x80000003] DDS_
 ↪DynamicData2TypePlugin_serialize:error copying CDR buffer (batching is not␣
 ↪supported)
ERROR [0x0101BE35,0x47B28D9B,0x87A311F6:0x80000004{Entity=DR,MessageKind=DATA}
 ↪|RECEIVE FROM 0x0101D676,0xE92E1D8E,0x91728D11:0x80000003]␣
 ↪PRESPsReaderQueue_evaluateSample:serialize failed
ERROR [0x0101BE35,0x47B28D9B,0x87A311F6:0x80000004{Entity=DR,MessageKind=DATA}
 ↪|RECEIVE FROM 0x0101D676,0xE92E1D8E,0x91728D11:0x80000003]␣
 ↪PRESPsReaderQueue_addQueueEntryToPolled:The sample couldn't be evaluated
```

With a ContentFilteredTopic, the problem only occurred when filtering was done on the *DataReader* side. The problem did not affect writer-side filtering.

[RTI Issue ID CORE-14539]

5.6.9 TopicQueries

[Major] Response to snapshot TopicQuery may have missed samples for instances that passed the TopicQuery filter

A response to a snapshot TopicQuery may not have included samples for an instance that passed the TopicQuery filter when the snapshot TopicQuery was received.

To understand the problem, assume a *Topic* on PatientData:

```
struct PatientData {
    @key long id;
    string name;
```

```
    string address;
};
```

Also assume that a *DataReader* creates a TopicQuery with the expression *id = 1234* and that a *DataWriter* with KEEP_LAST = 1 responding to the TopicQuery has a sample for that instance. For example:

```
{1234, 'Patient A', 'Street 345'}
```

When the *DataWriter* starts responding to the TopicQuery, it registers the last sequence number available in the *DataWriter* queue. This is the snapshot TopicQuery cutoff sequence number (SN). Then, the *DataWriter* starts answering the TopicQuery.

Before the *DataWriter* has a chance to send the sample for patient 1234, a hospital employee updates the patient's address as follows:

```
{1234, 'Patient A', 'Street 346'}
```

The new update will get an SN that is higher than the cutoff SN. As a result, the *DataReader* waiting for the TopicQuery response will not get any sample for patient 1234, which is wrong because the patient is still in the system.

This release has updated the TopicQuery implementation so that if the number of samples received for a TopicQuery is not equal to the number of samples published in response to the TopicQuery, the application is notified by setting a new flag DDS_INCOMPLETE_SNAPSHOT_TOPIC_QUERY on the last sample of the TopicQuery. This allows the application to detect incomplete snapshots and potentially reissue the TopicQuery.

In addition, for KEEP_LAST *DataWriter* configurations, the cutoff SN for a snapshot TopicQuery is re-evaluated if any sample is replaced (due to KEEP_LAST) while the TopicQuery is dispatched. This ensures more reliable data transmission.

For more information, see the Topic Queries chapter of the *RTI Connext Core Libraries User's Manual*.

[RTI Issue ID CORE-14439]

5.6.10 Request-Reply and Remote Procedure Calls

[Major] Python RPC API didn't support str arguments

Declaring an argument or a return type directly as a string (`str`) was not supported by the Python RPC API. This has been resolved, and the following example is now supported:

```
import rti.rpc as rpc
import rti.types as idl

@rpc.service
class MyService:
    @rpc.operation(
          parameter_annotations={"a": [idl.bound(10)], "return_": [idl.
↪bound(20)]}
     )
```

```
    def my_operation(self, a: str, b: str) -> str:
        pass
```

The service above is equivalent to the following service defined in IDL:

```
@service("DDS")
interface MyService {
     string<20> my_operation(string<10> a, string b);
};
```

[RTI Issue ID PY-177]

5.6.11 Logging

[Trivial] Error messages printed if reader responded to heartbeat from remote writer that was partially removed

This issue was fixed in release 7.3.0, but not documented at that time.

When a remote *DomainParticipant* is removed (for example, due to a partition change to an unmatching partition), the local *DomainParticipant* will remove the remote endpoints associated with the remote participant. Until the remote endpoints are fully removed, local *DataReaders* may still attempt to process and respond to heartbeats from a remote *DataWriter*. You may have encountered the following ERROR level message if this occurred:

```
ERROR [0x0101A69B,0xF74F3CBB,0x247AF7B3:0x00010187{Entity=DR,
 ↪MessageKind=HEARTBEAT}|RECEIVE FROM 0x01012628,0x6F49954A,
 ↪0x17AF8C82:0x00010182] PRESPsService_readerSampleListenerOnGetFreeCount:!
 ↪update liveliness for valid local reader remote writer matches
```

This error message has been removed for the cases where a remote *DataWriter* has been partially removed. You may now encounter the following REMOTE level log message instead, indicating that *Connext* has failed to look up the remote *DataWriter*:

```
REMOTE [0x010159FA,0xF14CAC35,0xD9959584:0x00010187{Entity=DR,
 ↪MessageKind=HEARTBEAT}|RECEIVE FROM 0x01015572,0xFE4893B3,
 ↪0x38A8F9A8:0x00010182] PRESPsService_
 ↪updateLocalReaderRemoteWriterMatchesWithAction:!goto WR pres psRemoteWriter
```

[RTI Issue ID CORE-14246]

[Trivial] Failure getting the Logger instance during instance creation overwrote the verbosity levels of all logging categories to random levels

When the Distributed Logger instance is being created (by calling `DistLogger::get_instance()`), *Connext* temporarily sets the verbosity of each logging category to `silent` to avoid logging messages while the creation is not completed. The verbosity is restored to the previous values once the creation finished.

If the internal function that caches the verbosity levels failed getting the `rti::config::Logger` instance, the variable that stored the previous verbosity levels was not populated and the `get_instance` operation continued without realizing the failure. Then, when the verbosities were restored to their "original values", they were overwritten with arbitrary values instead. As a result, after the failure, the verbosity level for each category was unintentionally changed to a random level.

Before modifying the verbosities, the following Distributed Logger error was printed:

```
DL Error: RTI_DL_DistLogger_maskSystemVerbosity: Unable to get RTI Logger
```

This issue is fixed. Now, if the caching of the verbosity levels fails, the `get_instance` method is notified and the execution is aborted.

Note that it is highly unlikely that the function that caches the verbosity levels fails.

[RTI Issue ID DISTLOG-241]

5.6.12 Dynamic Data

[Major] Incorrect DynamicData behavior involving empty structures

Creating a new DynamicData instance from an empty structure incorrectly failed. For example, consider the following Modern C++ code snippet:

```
using namespace dds::core::xtypes;

    StructType foo(
        "Foo", {}
    );

    StructType myFoo("Foo");
    DynamicData dynFoo(myFoo);
```

"Foo" is an empty structure. Its lack of members caused the constructor to fail with the following error:

```
DDS_DynamicData2_allocateMembers: Could not reserve buffer of 0 bytes for␣
↪values.
```

This problem affected all language APIs and has been fixed. DynamicData creation from an empty structure now succeeds.

The JSON-formatted string for a DynamicData sample was malformed if the sample had a base structure that was empty. For example, consider the following IDL:

```
struct EmptyStruct {};

struct StructWithEmptyBaseStruct : EmptyStruct {
    short x;
};
```

and the following Modern C++ code snippet:

```
DynamicData struct_with_empty_base_struct_dd(dynamic_type
↪<StructWithEmptyBaseStruct>::get());
    std::string str = rti::topic::to_string(
            struct_with_empty_base_struct_dd,
            PrintFormatProperty::Json());
```

The resulting `str` would look like `{,"x":0}`. The leading comma is considered a JSON syntax error. This problem has been fixed by removing this comma, so the `str` will now look like `{"x":0}`.

[RTI Issue ID CORE-14606]

[Major] DynamicData to and from JSON conversion did not include union discriminators

The discriminator value was not included in the JSON string produced by `DynamicData::to_json`. This could have caused a loss of information regarding what the discriminator was when there were multiple options. This behavior has been fixed, and the discriminator is now printed when needed.

[RTI Issue ID CORE-14259]

[Minor] DynamicData::get_member_info API returned incorrect member_id in some cases

When the `DynamicData::get_member_info API` was called for a member of a union that could be selected by multiple union discriminator values, the returned **DynamicDataMemberInfo::member_id** field was always set to the discriminator value of the first case label for that member. The correct behavior when the **member_id** was provided to the API in order to identify the member about which to get the information is to return the provided **member_id** in the **DynamicDataMemberInfo::member_id** field. For example, for the following union:

```
union Foo switch(short) {
case 1:
    long member1;
case2:
case3:
    long member2;
};
```

the following:

```
DDS_DynamicData_get_member_info(dynData, &info, 3, null);
```

returned **info.member_id** = 2, because 2 is the first label. Now, it correctly returns 3, the provided **member_id**.

[RTI Issue ID CORE-14583]

5.6.13 APIs (Modern C++ API)

[Major] Possible exception thrown due to incorrect DynamicData move constructor or assignment operator

In the Modern C++ API, the `DynamicData` move constructor and move-assignment operator didn't consider the case where the `DynamicData` object is "loaned." In this case, moving the object is not possible and must be copied. The application will have seen an error and exception thrown in that case.

For example, consider the following:

```
DynamicData sample = ...
LoanedDynamicData foo_member = sample.loan_value("foo");
foo_member.get() = DynamicData(...); // 1. move-assignment operator invoked␣
↪to assign the temporary rvalue DynamicData(...) to the loaned DynamicData␣
↪object
sample = DynamicData(...); // 2. move-assignment OK - objects are not loaned
```

This problem has been resolved. In the example above, the assignment makes a copy in (1) but continues to efficiently move the object in (2). Note that the move constructor and assignment operators are no longer `noexcept`, since a copy, which is now possible, is not `noexcept`.

[RTI Issue ID CORE-14581]

[Major] Modern C++ API was not enabled for x64Vx7SR0630llvm8.0.0.2_rtp

In *Connext* 7.3.0, the target package for `x64Vx7SR0630llvm8.0.0.2_rtp` did not include the Modern C++ API library, `libnddscpp2[d].so`.

[RTI Issue ID PLATFORMS-4417]

[Minor] find_member_by_id returned INVALID_INDEX in some cases

When using the Modern C++ API, calling `StructType::find_member_by_id()` may have returned INVALID_INDEX.

For example, the following code was susceptible to the problem:

```
MyStruct my_struct;
std::cout << dds::core::xtypes::StructType(static_cast<const␣
↪dds::core::xtypes::StructType &>(my_dynamic_data.type())).find_member_by_␣
↪id(100) << "\n";
```

whereas this code was not problematic:

```
MyStruct my_struct;
std::cout << static_cast<const dds::core::xtypes::StructType &>(my_dynamic_␣
↪data.type()).find_member_by_id(100) << "\n";
```

[RTI Issue ID CORE-14518]

[Trivial] SampleProcessor not included by rti/rti.hpp header

The header `rti/rti.hpp`, which is intended to include the full C++ API, didn't include `rti/sub/SampleProcessor.hpp`. Now, the `SampleProcessor` utility can be used by including `rti/rti.hpp`, `rti/sub/rtisub.hpp`, or `rti/sub/SampleProcessor.hpp`.

[RTI Issue ID CORE-14716]

5.6.14 APIs (Java)

[Major] Unnecessary memory operations when receiving bounded sequences

DataReaders receiving samples containing bounded sequences in the Java API may have incurred unnecessary memory operations. Receiving samples in these scenarios should now be more memory-efficient.

[RTI Issue ID CORE-14667]

5.6.15 APIs (Python)

[Major] Exception when printing or converting a sequence of octets to string

Trying to convert a user type containing sequences or arrays of octets to a string in the Python API may have caused a `UnicodeDecodeError`.

For example, given the following type in IDL:

```
struct MyType {
    sequence<octet> my_octets;
};
```

Or as defined in Python:

```
@idl.struct
class MyType:
my_octets: Sequence[idl.uint8] = field(default_factory=list)
```

The problem specifically affected the implementation of the `repr` methods of the types `dds.Int8Seq` and `dds.Uint8Seq`. These methods printed the elements as characters instead of integers.

Now, octet sequences are converted to strings correctly, and the elements are treated as integers instead of characters.

[RTI Issue ID PY-163]

[Major] Incorrect deserialization of nested union with unknown discriminator

When a `dds.DataReader` receives a data sample in which a union contains an unknown discriminator, the union object (`obj`) is expected to be initialized with the received discriminator and a `value` set to `None`. However, the `value` was not `None` when the union was nested (it was correctly set to `None` if the union was the top-level type).

[RTI Issue ID PY-182]

[Major] x64Linux Connext Python API not able to use libraries from $NDDSHOME/lib

On x64Linux, applications using the Connext Python API and relying on libraries in `$NDDSHOME/lib` to, for example, load an add-on library such as the *Security Plugins*, failed due to an undefined symbol issue due to ABI incompatibility. Note that normally the Python API doesn't load the native libraries from Nddshome.

[RTI Issue ID PY-168]

[Minor] QosProvider could not be directly created with QosProviderParams

A `QosProvider` could not be directly created with `QosProviderParams`; it required setting the parameters after creation.

The missing `init` method has been added and the following example code is now legal:

```
provider = dds.QosProvider(
    dds.QosProviderParams(
        url_profile=["file1.xml", "file2.xml"], ignore_user_profile=True
    )
)
```

[RTI Issue ID PY-171]

5.6.16 APIs (Multiple Languages)

[Major] DynamicData Errors when operating on unions with unknown discriminator values

A union's discriminator value may not select any member in the union. When this was the case, if the union was converted to a string with the `FooTypeSupport::data_to_string()` API (`rti::topic::to_string()` API in Modern C++) or printed using the `DynamicData::print()` API, the following errors were printed:

```
DDS_DynamicData2Visitor_visitMember:type not supported (DDS_TK_NULL)
DDS_DynamicData2Visitor_visitMember:!Error visiting union discriminator
DDS_DynamicData2Visitor_visitContainer: Error visiting MyUnion
DDS_DynamicData2_printI:!visit container
```

Now, the sample will be correctly printed with just the discriminator value and no associated member value.

The Modern C++ `DynamicData::get_discriminator()` API also failed to retrieve the discriminator value with an exception and the following errors:

```
DDS_DynamicData2UnionPlugin_getMemberInfo: Cannot find a member with id 0 in␣
↪type MyUnion
DDS_DynamicData2_get_member_info_by_index:!get_member_info
```

This fix also addresses RTI Issue IDs [CORE-14055] and [CORE-14443].

[RTI Issue ID CORE-14582]

[Minor] Creating a DomainParticipant with a negative domain ID was incorrectly allowed

The API Reference documentation for `create_participant` states that the `domainId` has **[range]** [>=0]. However, this range was not enforced, and *DomainParticipant* creation incorrectly succeeded if the `domainId` was a negative integer. Now, *DomainParticipant* creation will fail with this error message:

```
DDS_DomainParticipantFactory_create_participant_disabledI:BAD PARAMETER␣
↪FAILURE | domainId must be between 0 and 2147483647.
```

[RTI Issue ID CORE-5910]

5.6.17 XML Configuration

[Major] Could not use multiple <types> tags to create user data types

Consider the following XML file, which is used to create user data types:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="../xsd/rti_dds_profiles.xsd">
    <types>
        <struct name="Struct1">
            <member name="m1" type="short"/>
        </struct>
    </types>

    <types>
        <struct name="Struct2">
            <member name="m1" type="nonBasic" nonBasicTypeName="Struct1"/>
        </struct>
    </types>
</dds>
```

If you configured your `DomainParticipantFactoryQos` to have this XML file in your `profile.url_profile` value, and you attempted to call `DDS_DomainParticipantFactory_set_qos` on this QoS, this call would have incorrectly failed with the following error:

```
DDS_XMLTypeCode_reportNotFoundTypeSymbol:Parse error at line 11: type 'Struct1
↪' not found
```

This error happened because the second <types> tag was incorrectly treated as containing a set of type definitions that were independent from those contained in the first <types> tag.

Another consequence of this incorrect behavior was that only the first <types> tag was considered when looking up the typecode of a given structure. Consider the following XML file:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="../xsd/rti_dds_profiles.xsd">
    <types>
        <struct name="Struct1">
            <member name="m1" type="short"/>
        </struct>
    </types>

    <types>
        <struct name="Struct2">
            <member name="m1" type="short"/>
        </struct>
    </types>
</dds>
```

In this case, calling DDS_DomainParticipantFactory_set_qos would have succeeded, but then calling

```
DDS_DomainParticipantFactory_get_typecode_from_config(
        DDS_TheParticipantFactory,
        "Struct2");
```

would have incorrectly failed because Struct2 does not exist under the first <types> tag.

These problems have been fixed by treating the second <types> tag as an extension of the first one. This new behavior also works across multiple XML files; if the two <types> tags are in different XML files, then the second <types> tag will still be considered an extension of the first one.

[RTI Issue ID CORE-14680]

[Minor] Duplicate type names were allowed in an XML file for defining user data types

Consider the following XML file, which is used to create user data types:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="../xsd/rti_dds_profiles.xsd">
    <types>
        <struct name="Struct1">
            <member name="m1" type="short"/>
        </struct>
        <struct name="Struct1">
            <member name="m2" type="long"/>
        </struct>
    </types>
</dds>
```

If you configured your `DomainParticipantFactoryQos` to have this XML file in your `profile.url_profile` value, and you attempted to call `DDS_DomainParticipantFactory_set_qos` on this QoS, this call would have incorrectly succeeded when the duplicate `Struct1` names should have caused it to fail. A similar problem existed for `enum`, `bitset`, `valuetype`, `sparse_valuetype`, `union`, `typedef`, and `const` tags.

[RTI Issue ID CORE-14796]

[Trivial] DDS-prefixed enumerations caused confusion in XSD files

Before this release, the DDS-prefixed enumerations DDS_KEEP_LAST_HISTORY_QOS and DDS_KEEP_ALL_HISTORY_QOS were valid in the XSD files, as well as the standard enumerations, KEEP_LAST_HISTORY_QOS and KEEP_ALL_HISTORY_QOS. Having both sets of enumerations (with and without the DDS_ prefix) in the XSD files was confusing. In this release, RTI has updated the XSD files by removing the non-standard enumerations DDS_KEEP_LAST_HISTORY_QOS and DDS_KEEP_ALL_HISTORY_QOS and keeping the standard ones, KEEP_LAST_HISTORY_QOS and KEEP_ALL_HISTORY_QOS.

The primary goal of this change is to ensure that tools utilizing XSD files are aligned with standard DDS enumerations, thereby reducing potential confusion and maintaining consistency across applications. It's important to note that while these DDS-prefixed enumerations have been removed from the XSD files, they remain valid within the *Connext* XML parser. This means that existing configurations using these terms will continue to operate as expected.

However, anyone leveraging third-party tools for XML file validation might encounter warnings or errors if the removed DDS-prefixed enumerations are still in use. These tools may no longer recognize these enumerations as valid due to the enumerations' absence in the updated XSD files. We recommend reviewing and updating your XML configurations to use the standard enumeration terms to ensure seamless compatibility and validation across all tools and applications.

[RTI Issue ID CORE-14012]

5.6.18 Crashes

[Critical] Potential crash when creating new GuardCondition when low on resources

In the Java API, the creation of a new GuardCondition could have led to a crash. This only happened when there was an issue in allocating native resources.

[RTI Issue ID CORE-14544]

[Critical] Parsing an XML file with a root "types" tag that contained a "name" attribute led to a segmentation fault

When *Connext* parsed an XML file with a root "types" tag, and that tag contained a "name" attribute, a segmentation fault occurred. The use of the name attribute in a root types tag is not permitted by the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3, so this error would only occur with non-conforming XML files. Other non-conforming XML files, such as those with invalid root tags that contained elements with the name attribute, also caused this issue.

[RTI Issue ID CORE-14422]

[Critical] Parsing of malformed RTPS packages may have led to bus error in some platforms

Parsing malformed RTPS packages (with or without *RTI Security Plugins*) may have caused a bus error on QNX platforms running on an armv7 processor.

[RTI Issue ID CORE-14617]

[Critical] Potential race conditions in construction or deletion of Requesters and Repliers (C, Traditional C++, C#, and Python)

A race condition may have caused crashes or exceptions when constructing or deleting Requesters and Repliers in the C, Traditional C++, C#, and Python APIs.

These issues were previously fixed as RTI Issue IDs [REQREPLY-127] and [REQREPLY-132] for the Modern C++ and Java APIs.

[RTI Issue ID REQREPLY-133]

[Critical] Possible crash when creating dds.DynamicData.Topic with a listener

A crash in the Python API may have occurred while creating a `dds.DynamicData.Topic` with a listener. The problem did not occur if the listener was set *after* creating the topic with `set_listener()`. This problem did not affect IDL-based topics (`dds.Topic`).

[RTI Issue ID PY-159]

[Critical] Possible crash after a failure to create Monitoring Library 2.0 object

If *RTI Monitoring Library* 2.0 was enabled in your application, and something failed while creating the library object (for example, if the *Monitoring Library* 2.0 internal DDS entities could not be created because of an inconsistent QoS configuration), the system may have crashed.

The crash happened during clean-up after the failure, because some improperly initialized pointers were accessed. Before the crash, you may have seen errors in the `RTI_Monitoring_create()` function.

To prevent this crash, all the pointers are now initialized to NULL before calling `RTI_Monitoring_create()`. Note that, depending on how your platform handles pointers initialization, this might or might not be a problem.

[RTI Issue ID MONITOR-668]

[Critical] Possible segmentation fault while enabling a DataWriter that enables batching

Consider the following scenario:

- You are using the Java API.
- The `DataWriterQos` has `batch.enable` set to true.

Attempting to enable a *DataWriter* with that QoS would occasionally fail with a segmentation fault in the internal function `PRESTypePluginDefaultEndpointData_calculateBatchBufferSize()`. This problem only affected releases 7.0.0 to 7.3.0.

[RTI Issue ID CORE-14659]

[Critical] Segmentation fault when copying invalid samples and using Zero Copy transfer over shared memory transport

When reading or taking samples from a *DataReader's* queue, the samples can either be loaned or copied from the queue, depending on which version of the `read` or `take` API is used and how it is used. When a *DataReader* that supported receiving samples over the *Zero Copy transfer over shared memory* transport used the `read` or `take` APIs that copied samples from the *DataReader's* queue (instead of loaning them), the application crashed if any of the samples had the `SampleInfo::valid_data` flag set to false. The only safe APIs for copying samples from the *DataReader's* queue were `FooDataReader::read_next_sample()` and `FooDataReader::take_next_sample()` because those APIs skip samples with the `valid_data` flag set to false.

[RTI Issue ID CORE-14442]

[Critical] Possible crash when calling DDS_DataWriter_get_matched_subscriptions() or DDS_DataReader_get_matched_publications() on architectures with a weak memory model

On architectures that use a weak memory model, such as Arm-based architectures, the application may have crashed when calling `DDS_DataWriter_get_matched_subscriptions()` or `DDS_DataReader_get_matched_publications()` (or their C++ equivalents) at the same time that a new matching entity was discovered. The crash happened in some rare cases due to a reordering of the memory storages. Now the mentioned APIs can be called at the same time that a new matching entity is discovered.

[RTI Issue ID CORE-14743]

[Critical] Possible crash when using QosProvider to load XML file with syntax `file:///` on Windows systems

This issue was fixed in 7.3.0, but not documented at that time.

When using a QosProvider to load profiles from a file, the documented syntax for specifying the file's URL is to use three slashes: `file:///`. However, using the `file:///` syntax caused a crash on Windows systems. The workaround was to use two slashes, `file://`.

To address this, both `file:///` and `file://` are now valid.

[RTI Issue ID CORE-14603]

[Critical] Crash on DDS_DomainParticipantFactory_finalize_instance() when using XML-Based Application Creation

When using XML-Based Application Creation with multiple registered types, `DDS_DomainParticipantFactory_finalize_instance()` could have crashed the JVM.

[RTI Issue ID CORE-14912]

[Critical] Parsing of malformed RTPS packets may have led to bus error on some platforms

Parsing malformed RTPS packets (with or without *RTI Security Plugins*) may have caused a bus error on QNX platforms running on an armv7 processor (this may not be the only platform affected). This was fixed and a malformed RTPS message is now dropped before causing the bus error.

[RTI Issue ID CORE-14549]

[Critical] Crash if element in initial_peers longer than 72 characters added

If a *DomainParticipant* had an element in the `initial_peers` longer than 72 characters, or if a peer longer than 72 characters was added using `DDS_DomainParticipant_add_peer()`, the program would crash. This problem has been resolved, and now peers up to 210 characters in length can be added. If a peer is added that is longer than 210 characters, the program will not crash, but the *DomainParticipant* will print the following set of errors:

```
DDS_DiscoveryQosPolicy_parseParticipantPeerDescriptor: locator string too␣
↪long:␣
↪BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
ERROR [0x010134D9,0x574C97F0,0xE8BB2C8C:0x000001C1{Domain=0}|ADD_PEER␣
↪BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
↪DDS_DiscoveryQosPolicy_parsePeerDescriptorString:peer descriptor  format is␣
↪unrecognized, string value =
↪"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
↪"
ERROR [0x010134D9,0x574C97F0,0xE8BB2C8C:0x000001C1{Domain=0}|ADD_PEER␣
↪BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
↪DDS_DomainParticipantDiscovery_add_peer:invalid peer descriptor
```

[RTI Issue ID CORE-14970]

[Major] Potential crash upon serialization, deserialization, or program shutdown when running out of memory

Serialization, deserialization, and program shutdown could have crashed if the program ran out of memory. The problem affected all APIs except Java. This situation will now fail gracefully, without crashing.

[RTI Issue ID CORE-14656]

[Major] Possible crash when closing a logger device while it was being used

Due to a concurrency issue in the *Connext* logging infrastructure, there was the small possibility of a crash in an application when a logger device was closed at the same time it was being used for logging a message.

[RTI Issue ID CORE-10546]

[Major] Crash during custom transport load when running out of memory

A *Connext* application could crash if it ran out of memory while loading a custom transport plugin. Now the load fails gracefully when running out of memory.

[RTI Issue ID CORE-14706]

[Major] Precondition failure or segmentation fault when nonBasicTypeName is the name of a module

When attempting to load an XML file used for creating user data types, a precondition failure or segmentation fault occurred if the value of `nonBasicTypeName` was the name of a module. For example, consider this invalid XML file:

```
<types>
<module name="tc2idl">
  <struct name= "TestData">
    <member name="aWStringSequenceSequence" type="nonBasic" ␣
↪nonBasicTypeName= "tc2idl" arrayDimensions="3,4"/>
  </struct>
</module>
</types>
```

When attempting to load this XML file using the debug libraries for a 64-bit architecture, you would incorrectly get the following precondition error:

```
TypeCodeObject.c:140:RTI0x3000035:!precondition: "self == ((void *)0)"
```

This precondition error occurred because `tc2idl` is the name of a module, not the name of a struct, enum, bitset, typedef, valuetype, sparse_valuetype, or union.

When attempting to load this XML file using the release or debug libraries for a 32-bit architecture, you would get a segmentation fault in the function `RTICdrTypeCode_hasCdrRepresentation`.

When attempting to load this XML file using the release libraries for a 64-bit architecture, you would correctly get the following error:

```
DDS_XMLTypeCode_validateMemberTypeSymbol:Parse error at line 6: type 'tc2idl'␣
↪is not a struct, enum, bitset, typedef, valuetype, sparse_valuetype or union
```

Now, you will get this error regardless of release or debug libraries and regardless of 32-bit or 64-bit architecture.

[RTI Issue ID CORE-14779]

5.6.19 Hangs

[Critical] String built-in type unregister_type API blocked associated DomainParticipant

The `unregister_type` for the String built-in type did not work properly; it failed and blocked the *DomainParticipant* instead of unregistering the type.

[RTI Issue ID CORE-14543]

[Critical] Potential deadlock when using SHMEM transport on Linux, macOS, and LynxOS systems and killing one application

Consider the following scenario:

- Three applications using the the SHMEM transport are communicating with each other.
- One application is ungracefully killed while creating a *DomainParticipant*.

If the killing of the application occurred at just the wrong moment, then the other applications would stop communicating with each other because they were in a deadlock. The deadlock would persist until you cleaned the shared memory resources. In addition, any new *DomainParticipants* that attempted to use the same shared memory resources as the killed application would hang during *DomainParticipant* creation.

The problem could occur in Linux®, macOS®, and LynxOS® systems.

Now, instead of entering a deadlock, *Connext* will print a log message like this one at the WARNING level and the PLATFORM category:

```
RTIOsapiSharedMemorySemMutex_attach_os:FAILED TO ATTACH | Semaphore set␣
↪identifier 0x12345678 has not been initialized. Consider cleaning this␣
↪semaphore if this warning occurs multiple times for this semaphore across␣
↪all applications on this machine.
```

The semaphore set identifier corresponds to the `semid` column of the output of `ipcs -s` on Linux:

```
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x0010000c 305419896  username   666        1
```

[RTI Issue ID CORE-14848]

[Critical] Potential deadlock when creating entities in parallel that use Zero Copy transfer over shared memory transport

Consider the following scenario:

- A *DomainParticipant* is using the *Zero Copy transfer over shared memory transport*.
- One thread is trying to create a *DataWriter* or *DataReader* belonging to that *DomainParticipant*.
- Another thread is either:
  - trying to create a *DataWriter* or *DataReader* from a callback function of a builtin Topic *DataReader* belonging to that *DomainParticipant*, or
  - using the Request-Reply API and trying to create a Requester or Replier belonging to that *DomainParticipant*.

These two threads may have entered a deadlock while trying to create their entities.

[RTI Issue ID CORE-14820]

[Major] Running out of memory during DomainParticipant creation caused DomainParticipantFactory finalization to hang

During *DomainParticipant* creation, if the internal function `REDAWorkerFactory_createWorker()` runs out of memory and prints this error:

```
REDAWorkerFactory_createWorker:no space on heap for array with 1024 elements␣
↪of size 8 bytes
```

then *DomainParticipant* creation will fail. The problem was that when you later tried to finalize the *DomainParticipantFactory*, this operation would hang and repeatedly print these errors:

```
REDAWorkerFactory_destroyWorkerEx:!take mutex
```

This problem only affected versions 7.0.0 to 7.3.0.

[RTI Issue ID CORE-14962]

5.6.20 Memory Leaks/Growth

[Critical] Parsing XML configuration with large tokens caused uncontrolled resource consumption

If you added a very large token (greater than 1 MB) to an XML file parsed by *Connext*, it could have caused uncontrolled resource consumption since the underlying XML parser would frequently reallocate memory to store the token. This problem has been addressed by upgrading **expat** to version 2.6.2.

[RTI Issue ID CORE-14529]

[Critical] Potential unbounded memory growth when using DNS tracking

Connext allows adding peers based on a hostname instead of an IP address. When the `dns_tracker_polling_period` is set, *Connext* attempts to update the address resolution of these hostnames at this period.

It was possible to experience an unbounded memory growth when using `initial_peers` that had overlapping sets of hostnames and IP addresses. For example, consider a *DomainParticipant* that was using the following `initial_peers: {[1]@udpv4://hostname, [4]@udpv4://hostname, [2]@udpv4://127.0.0.1, [3]@udpv4://127.0.0.1, [4]@udpv4://127.0.0.1}`. If `hostname` resolved to `127.0.0.1`, the *DomainParticipant* would duplicate the entry for `[4]@udpv4://127.0.0.1` when updating the DNS resolution. This would also result in duplicate discovery traffic sent to that locator. Now the memory leak and duplicate discovery traffic no longer occur.

[RTI Issue ID CORE-14151]

[Critical] Parsing DTD with many nested/recursive entities could cause uncontrolled resource consumption

A Document Type Definition (DTD) with several nested or recursive entities could have caused uncontrolled resource consumption. This could have occurred when the DTD was parsed by **expat** (third-party software used by *Connext*). This problem has been addressed by upgrading **expat** to version 2.6.2.

[RTI Issue ID CORE-14576]

[Major] Memory leak when parsing malformed XML defined types

A memory leak could be seen when trying to parse a malformed XML file filled with types.

[RTI Issue ID CORE-14662]

[Minor] Possible memory leak when initializing and finalizing the DomainParticipantFactory concurrently

If one thread was initializing the *DomainParticipantFactory* (e.g., by calling `DDSDomainParticipantFactory::get_instance()` in traditional C++) and another thread was finalizing the DomainParticipantFactory (e.g., by calling `DDSDomainParticipantFactory::finalize_instance()` in traditional C++) at the same time, then a memory leak in the internal functions `RTIOsapiContext_associateThread` and `RTIOsapiContextSupport_assertContextTss` was possible due to a race condition. This problem did not lead to unbounded memory growth and only affected releases 6.1.0 to 7.4.0.

[RTI Issue ID CORE-14972]

[Minor] Memory leak when parsing XML QoS profile with unknown base_name

When attempting to load an XML QoS profile, a memory leak occurred if the value of `base_name` was a non-existent QoS profile. For example, consider this invalid XML QoS profile:

```
<qos_library name="MyLibrary">
  <qos_profile name="MyProfile" base_name="BuiltinQosLib::Gengric.
↪StrictReliable">
  </qos_profile>
</qos_library>
```

When attempting to load this XML file, you would correctly get the following error because of the typo in `Gengric`:

```
RTIXMLObject_initialize:Base object 'BuiltinQosLib::Gengric.StrictReliable'␣
↪not found
```

However, you would also incorrectly get a memory leak in the internal function `RTIXMLObject_initialize`.

[RTI Issue ID CORE-14918]

[Minor] Memory leak during DomainParticipant creation when running out of memory trying to create the event thread

If the internal function `RTIEventActiveGenerator_new` ran out of memory, then *DomainParticipant* creation would fail with a memory leak. Here is one example error message, along with a valgrind result:

```
COMMENDActiveFacade_new:!create eventGenerator

==26292== 856 (480 direct, 376 indirect) bytes in 1 blocks are definitely␣
↪lost in loss record 6 of 6
==26292==    at 0x4C33B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-
↪amd64-linux.so)
==26292==    by 0x12EACB0: RTIOsapiHeap_reallocateMemoryInternal (heap.c:831)
==26292==    by 0x1BEF10D: RTIEventActiveGenerator_new (ActiveGenerator.c:712)
```

The leak would only happen if memory was already exhausted, so this problem did not lead to unbounded memory growth.

[RTI Issue ID CORE-14878]

5.6.21 Data Corruption

[Major] Parsing of malformed DATA_FRAG submessage may have led to unexpected application behavior

Parsing a malformed DATA_FRAG submessage may have led to unexpected application behavior. The impact of this issue varied depending on the libraries used:

- For debug libraries, the following precondition error could be seen: `!precondition: "numFragmentsIn == 0"`.
- For release libraries, fragments in the DATA_FRAG message would not be considered received and therefore would be unnecessarily requested through NACKs. Eventually, this could stop the application from receiving new samples and to request the same fragments indefinitely.

[RTI Issue ID CORE-14548]

5.6.22 OMG Specification Compliance

[Major] Not possible to specify how to deserialize a sample with a discriminator that does not select a member or an unknown enum

Previously, when the properties `dds.sample_assignability.accept_unknown_enum_value` or `dds.sample_assignability.accept_unknown_union_discriminator` were used, they didn't modify the behavior of the `from_cdr_buffer` APIs. Now, instead of using these properties, you must set the behavior via the Extensible Types compliance mask, which correctly modifies the behavior of the `from_cdr_buffer` APIs as follows:

- To enable what was previously the property `dds.sample_assignability.accept_unknown_enum_value`, you would now set the bit `NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_ENUM_VALUE_BIT`, whose value is `0x00000010`, in the Extensible Types compliance mask. When this bit is set, samples with an unknown enumerator will be accepted when deserializing or calling `from_cdr_buffer`.
- To enable what was previously the property `dds.sample_assignability.accept_unknown_union_discriminator` depends on the value you were assigning to it; `from_cdr_buffer` will also be affected by the value of these bits:
  - 0 - Reject samples with an unknown discriminator: you don't need to set any bits.
  - 1 - Accept samples with an unknown discriminator and assign the default discriminator value: you need to set the bit NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT, whose value is `0x00000020`, to accept the sample, and set the bit NDDS_CONFIG_XTYPES_SELECT_DEFAULT_DISCRIMINATOR_BIT, whose value is `0x00000040`, to select the default discriminator value. If this bit is left unset and the NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT bit is set, then the sample will be accepted and the unknown union discriminator value will be preserved.
  - 2 - Accept samples with an unknown discriminator: set the bit `NDDS_CONFIG_XTYPES_ACCEPT_UNKNOWN_DISCRIMINATOR_BIT`, whose value is `0x00000020`, in the Extensible Types compliance mask.

With the new Extensible Types compliance mask (added in release 7.3.0), the properties `dds.sample_assignability.accept_unknown_enum_value` and `dds.sample_assignability.accept_unknown_union_discriminator` have now been deprecated, which means they will be removed from a future release.

Now that you can configure these deserialization aspects of the `from_cdr_buffer`, the default behavior of the `from_cdr_buffer` APIs has changed to reject samples with unknown union discriminators and unknown enum values. If you want the previous behavior, see the Migration Guide.

[RTI Issue ID CORE-14532]

5.6.23 Interoperability

[Critical] Incorrect handling of RTPS messages with submessages from different participants

When an RTPS message that contained submessages from multiple participants was received, *Connext* incorrectly treated each submessage as though it were from the participant whose GUID prefix was in the RTPS header. *Connext* does not send RTPS messages with submessages from different participants, but other DDS vendors may do this, which would have led to various communication issues and a lack of interoperability.

[RTI Issue ID CORE-8336]

[Critical] RTPS checksums incorrect for multiple RTPS messages received in a single datagram

The value of RTPS checksums calculated upon message reception was incorrect when multiple RTPS messages were present in a single datagram. This led to discarding all messages in the datagram and in severe cases could prevent communication. This issue affected all available RTPS checksum mechanisms.

[RTI Issue ID CORE-14684]

5.6.24 Vulnerabilities

[Critical] Stack buffer write overflow while parsing malicious license file

An out-of-bounds write on the stack could occur while parsing a malicious license file.

User Impact without Security

A vulnerability in the *Connext* application could have resulted in the following:

- Stack buffer overflow while parsing a malicious license file.
- Exploitable by overwriting the license file on the file system with a malicious license file.
- Potential impact on integrity of *Connext* application.
- CVSS Base Score: 6.1 MEDIUM
- CVSS v3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:L

User Impact with Security

Same impact as "User Impact without Security" above.

[RTI Issue ID CORE-14875]

[Critical] Stack buffer write overflow while parsing malicious XML types document

An out-of-bounds write on the stack could occur while parsing a malicious XML types document.

User Impact without Security

A vulnerability in the Core Libraries affected all products that load types via XML, and could have resulted in the following:

- Stack buffer overflow while parsing a malicious XML types document.
- Exploitable by changing an XML configuration file on the file system.
- Potential impact on the integrity of the application(s) using the XML types document.
- Potential crash in the application.
- CVSS Base Score: 7.1 HIGH
- CVSS 3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

User Impact with Security

Same impact as "User Impact without Security" above.

[RTI Issue ID CORE-14872]

[Critical] Stack buffer write overflow while parsing a malicious XML types document

An out-of-bounds write on the stack could occur while parsing a malicious XML types document.

User Impact without Security

A vulnerability in the Core Libraries affected all products that load types via XML, and could have resulted in the following:

- Stack buffer overflow while parsing a malicious XML types document.
- Exploitable by changing an XML configuration file on the file system.
- Potential impact on the integrity of the application(s) using the XML types document. Such applications could include *RTI Routing Service*.
- Potential crash in the application.
- In the case of *Routing Service*, the vulnerability could potentially be triggered through the remote administration command load, but a successful attack would require a malicious XML include file to already exist in the system, so the "Attack Vector" score is still "Local".
- CVSS Base Score: 7.1 HIGH
- CVSS 3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

User Impact with Security

Same impact as "User Impact without Security" above.

[RTI Issue ID CORE-14871]

5.6.25 Other

[Critical] Data races on architectures with weak memory model

Several data races on systems with a weak memory architecture, such as ARM and PowerPC, have been fixed in this release. Some of these data races could have led to undefined behavior or crashes in different parts of *Connext*.

[RTI Issue ID CORE-14893]

[Major] Missing implementation of RTI_DL_DistLogger_get_version(), RTI_DL_DistLogger_get_api_build_version(), and RTI_DL_DistLogger_get_api_version_string()

The implementation of the APIs `RTI_DL_DistLogger_get_version()`, `RTI_DL_DistLogger_get_api_build_version()`, and `RTI_DL_DistLogger_get_api_version_string()` was not compiled, so using them resulted in a "function declared but not defined" failure at compile time. The APIs can now be used as expected.

[RTI Issue ID DISTLOG-243]

[Major] Incorrect update of DataWriter's pushed_fragment_count protocol statistic under some conditions

Connext determines the need for fragmenting a sample using the configured `message_size_max` across all installed transports. If at least one transport requires fragmentation, *Connext* will fragment the sample, even if the transport ultimately used for transmission does not require it. In such a scenario (where the used transport does not require fragmentation), *Connext* optimizes communication by grouping all the fragments in a single `DATA_FRAG` sub-message.

The `DataWriterProtocolStatus::pushed_fragment_count` statistic was not correctly updated in this scenario. The count was increased using the number of fragments added to the `DATA_FRAG` sub-message instead of the actual number of `DATA_FRAG` sub-messages sent on the wire.

This issue occurred for asynchronous *DataWriters* communicating with best-effort *DataReaders* when the Flow Controller `bytes_per_token` and `max_tokens` configuration required fragmenting the sample anyway.

Besides the scenario described here, the `pushed_fragment_count` also did not update properly for synchronous best-effort communications when fragmentation was required and a ContentFilteredTopic was involved.

The `pushed_fragment_count` statistics are now accurate.

[RTI Issue ID CORE-14737]

[Minor] IDL annotations printed after typdef specifier

When using `to_string` on IDL types where annotations (such as `@default`) and a `typedef` were specified, those annotations were printed after the `typedef` specifier. This behavior has been changed to print the `typdef` specifier after the necessary annotations.

[RTI Issue ID CORE-14857]

Chapter 6

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.

This section includes:

6.1 Known Issues with Discovery

6.1.1 SPDP2

The following known issues apply to the Simple Participant Discovery Protocol 2.0, which is an alternative version of the Simple Participant Discovery Protocol, designed for decreased bandwidth usage and improved reliability. See Simple Participant Discovery 2.0, in the RTI Connext Core Libraries User's Manual for more information.

Features under future consideration for SPDP2

Note: RTI does not guarantee the following features for any release or timeline. If any of these enhancements is of interest to you, please provide that feedback through your account team.

The following features, which are not currently supported, are being considered for SPDP2 in future releases:

- Use of SPDP2 with custom security plugins (for example, those implemented with the Security Plugins SDK *(RTI Security Plugins SDK)*). Only the *Builtin Security Plugins* and the *Lightweight Builtin Security Plugins* are supported in combination with SPDP2.
- SPDP and SPDP2 compatibility mode. The compatibility mode will allow some *DomainParticipants* to simultaneously communicate with *DomainParticipants* that are using SPDP and SPDP2. *DomainParticipants* that are using the compatibility mode will be able to communicate with *DomainParticipants* that

are using SPDP and other *DomainParticipants* that are using SPDP2. For now, you can use RTI Routing Service to achieve this communication.

- Improved configuration update behavior. Currently, when a *DomainParticipant* changes its configuration (partition, locators, etc.), it sends out:
  - If SPDP is enabled: a single Data(p) to all peers (matched or potential).
  - If SPDP2 is enabled: a single reliable message to matched peers, a single bootstrap message to unmatched initial peers. RTI will add an option to send multiple Data(p)s/bootstrap messages, since these messages are sent best-effort and can get lost, delaying configuration change updates in remote participants until the next periodic message.

[RTI Issue IDs CORE-12929, CORE-13884, and CORE-12930]

Participants using SPDP2 and allow_unauthenticated_participants fail to communicate if only one participant fails authentication

DomainParticipants using SPDP2 with **allow_unauthenticated_participants** set to TRUE fail to communicate if both participants are using security and only one participant fails authentication.

[RTI Issue ID SEC-2348]

6.1.2 TypeObject v2 not serialized correctly on big endian platforms

On big endian platforms, the TypeLookupService reply *DataWriter* serializes the requested TypeObject v2 objects in big endian. The OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.4 mandates that TypeObject v2 objects must always be serialized in little endian. *Connext* applications will successfully deserialize these samples, but it is not guaranteed that other DDS vendors will.

[RTI Issue ID CORE-16510]

6.2 Known Issues with Serialization and Deserialization

6.2.1 Some parameters cannot be received multiple times within same SPDP sample

The OMG Real-Time Publish-Subscribe (RTPS) specification, version 2.5 allows in general that "The ParameterList may contain multiple Parameters with the same value for the parameterId." *RTI Connext*, however, does not support receiving the following parameterId values multiple times within the same Simple Participant Discovery Protocol (SPDP) discovery sample:

- PID_USER_DATA
- PID_PROPERTY_LIST
- PID_ENTITY_NAME
- PID_ROLE_NAME

- PID_PARTITION
- PID_DOMAIN_TAG
- PID_IDENTITY_TOKEN
- PID_PERMISSIONS_TOKEN
- PID_TRANSPORT_INFO_LIST

[RTI Issue ID CORE-13680]

6.2.2 Connext not compliant with Extended CDR encoding version 2 for types containing arrays and sequences of non-primitive types

By default, *Connext* is not compliant with Extended CDR encoding version 2 for types containing arrays and sequences of non-primitive types. To configure *Connext* to be compliant, you have a few options, described in Extended CDR (encoding version 2), in the *Extensible Types Guide*.

6.3 Known Issues with Usability

6.3.1 Cannot open USER_QOS_PROFILES.xml in rti_workspace/examples from Visual Studio

When trying to open the **USER_QOS_PROFILES.xml** file from the resource folder of one of the provided examples, you may see the following error:

```
Could not find file : C:\Users\<user>\Documents\rti_workspace\5.3.0\examples\
↪connext_dds\c\<example>\win32\USER_QOS_PROFILES.xml
```

The problem is that the Visual Studio project is looking for the file in a wrong location (win32 folder).

You can open the file manually from here:

```
C:\Users\<user>\Documents\rti_workspace\5.3.0\examples\connext_dds\c\<example>
↪\USER_QOS_PROFILES.xml
```

This issue does not affect the functionality of the example.

[RTI Issue ID CODEGENII-743]

6.3.2 DataWriter's Listener callback on_application_acknowledgment() not triggered by late-joining DataReaders

The *DataWriter's* listener callback **on_application_acknowledgment()** may not be triggered by late-joining *DataReaders* for a sample after the sample has been application-level acknowledged by all live *DataReaders* (no late-joiners).

If your application requires acknowledgment of message receipt by late-joiners, use the Request/Reply communication pattern with an Acknowledgment type (see the chapter "Introduction to the Request-Reply Communication Pattern," in the *Core Libraries User's Manual*).

[RTI Issue ID CORE-5181]

6.3.3 HighThroughput and AutoTuning built-in QoS Profiles may cause communication failure when writing small samples

If you inherit from either the **BuiltinQosLibExp::Generic.StrictReliable.HighThroughput** or the **BuiltinQosLibExp::Generic.AutoTuning** built-in QoS profiles, your *DataWriters* and *DataReaders* will fail to communicate if you are writing small samples.

In *Connext* 5.1.0, if you wrote samples that were smaller than 384 bytes, you would run into this problem. In version 5.2.0 onward, you might experience this problem when writing samples that are smaller than 120 bytes.

This communication failure is due to an interaction between the batching QoS settings in the **Generic.HighThroughput** profile and the *DataReader*'s **max_samples** resource limit, set in the **BuiltinQosLibExp::Generic.StrictReliable** profile. The size of the batches that the *DataWriter* writes are limited to 30,720 bytes (see max_data_bytes). This means that if you are writing samples that are smaller than 30,720/**max_samples** bytes, each batch will have more than **max_samples** samples in it. The *DataReader* cannot handle a batch with more than **max_samples** samples and the batch will be dropped.

There are a number of ways to fix this problem, the most straightforward of which is to overwrite the *DataReader's* **max_samples** resource limit. In your own QoS profile, use a higher value that accommodates the number of samples that will be sent in each batch. (Simply divide 30,720 by the size of your samples).

[RTI Issue ID CORE-6411]

6.3.4 Memory leak if Foo:initialize() called twice

Calling **Foo:initialize()** more than once will cause a memory leak.

[RTI Issue ID CORE-7678]

6.3.5 Wrong error code after timeout on write() from Asynchronous Publisher

When using an asynchronous publisher, if **write()** times out, it will mistakenly return DDS_RETCODE_ERROR instead of the correct code, DDS_RETCODE_TIMEOUT.

[RTI Issue ID CORE-2016, Bug # 11362]

6.3.6 Type Consistency enforcement disabled for structs with more than 10000 members

TypeObjects cannot be created from structs with more than 10000 members. Applications that publish or subscribe to such types may see errors like the following:

```
RTICdrStream_serializeNonPrimitiveSequence:sequence length (10005) exceeds␣
↪maximum (10000)
RTICdrTypeObjectTypeLibraryElement_getTypeId:serialization error: Type
RTICdrTypeObject_fillType:!get TypeId
RTICdrTypeObject_assertTypeFromTypeCode:!create Structure Type
RTICdrTypeObject_createFromTypeCode:!create TypeObject
```

When the TypeObject can't be serialized, the type compatibility check between a reader and a writer falls back to exact type-name matching.

See the section "Verifying Type Consistency: Type Assignability" in the *RTI Connext Core Libraries Extensible Types Guide* for more information.

[RTI Issue ID CORE-8158]

6.3.7 Escaping special characters in regular/filter expressions not supported in some cases

Escaping special characters is not supported in expressions when using the following features:

- Partitions
- MultiChannel

Every occurrence of a backslash (`\`) will be considered its own character and not a way to escape the character that follows. For example: `A\?` does not match `A?` because the first expression is considered an expression with three characters.

[RTI Issue ID CORE-11858]

6.3.8 Non-English Unicode (UTF-8) file names and paths not supported

Connext cannot open files that contain non-English Unicode (UTF-8) characters in their name or path. For example, if the NDDS_QOS_PROFILES environment variable is set to point to a file with non-standard characters in the name, you will get an error similar to this one:

```
ERROR LC:Discovery| DDS_QosProvider_load_profiles_from_url_groupI:ERROR:␣
↪opening profiles group files 'rti_workspace/7.3.0/examples/connext_dds/c/
↪hello_UTF/??.xml '
```

This issue also affects files that *Connext* writes to, such as when logging is configured to write to a file.

[RTI Issue ID CORE-14901]

6.4 Known Issues with Code Generation

6.4.1 Examples and generated code for Visual Studio 2017 and later may not compile (Error MSB8036)

The examples provided with *Connext* and the code generated for Visual Studio 2017 and later will not compile out of the box if the Windows SDK version installed is not a specific number like 10.0.15063.0. If that happens, you will see the compilation error MSB8036. To compile these projects, select an installed version of Windows SDK from the Project menu -> Retarget solution.

Another option is to set the enviroment variable RTI_VS_WINDOWS_TARGET_PLATFORM_VERSION to the SDK version number. For example, set RTI_VS_WINDOWS_TARGET_PLATFORM_VERSION to 10.0.16299.0. (Note: the environment variable will not work if you have already retargeted the project via the Project menu.)

For further details, see the Windows chapter of the *Core Libraries Platform Notes*.

[RTI Issue ID CODEGENII-800]

6.5 Known Issues with Instance Lifecycle

6.5.1 RECOVER_INSTANCE_STATE_CONSISTENCY setting not fully supported by RTI Infrastructure Services

The RECOVER_INSTANCE_STATE_CONSISTENCY option in the **instance_state_consistency_kind** field, in the RELIABILITY QoS policy, is not fully supported by the *RTI Infrastructure Services* products.

RTI Routing Service inputs cannot route instance state transitions from NOT_ALIVE_NO_WRITERS to ALIVE after regaining liveliness with a *DataWriter*. However, a *Routing Service* output *DataWriter* can be configured to use the RECOVER_INSTANCE_STATE_CONSISTENCY setting and respond to matching *DataReaders* if they request instance state updates after a reconnection.

Persistence Service, *Queuing Service*, *Recording Service*, and *Replay Service* do not support being configured with the RECOVER_INSTANCE_STATE_CONSISTENCY setting, since they do not support storing or publishing ALIVE instance state transitions with no associated data.

[RTI Issue ID CORE-13337]

6.5.2 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]

6.5.3 instance_state_consistency_kind QoS cannot be modified before containing entity is enabled

The **instance_state_consistency_kind** field in the RELIABILITY QoS policy cannot be modified once the containing DDS entity is created, even if the containing entity is created disabled. Trying to modify the QoS setting in this case will result in the `set_qos` operation returning `DDS_RETCODE_IMMUTABLE_POLICY` and an error message being logged.

[RTI Issue ID CORE-13349]

6.6 Known Issues with Reliability

6.6.1 DataReaders with different reliability kinds under Subscriber with GROUP_PRESENTATION_QOS may cause communication failure

Creating a *Subscriber* with **PresentationQosPolicy.access_scope** GROUP_PRESENTATION_QOS and then creating *DataReaders* with different **ReliabilityQosPolicy.kind** values creates the potential for situations in which those *DataReaders* will not receive any data.

One such situation is when the *DataReaders* are discovered as late-joiners. In this case, samples are never delivered to the *DataReaders*. A workaround for this issue is to set the **AvailabilityQosPolicy.max_data_availabilty_waiting_time** to a finite value for each *DataReader*.

[RTI Issue ID CORE-7284]

6.7 Known Issues with Content Filters and Query Conditions

6.7.1 Writer-side filtering may cause missed deadline

If you are using a ContentFilteredTopic and you set the Deadline QosPolicy, the deadline may be missed due to filtering by a *DataWriter*.

[RTI Issue ID CORE-1634, Bug # 10765]

6.7.2 filter_sample_* statistics in DDS_DataWriterProtocolStatus not updated correctly

The **filter_sample_*** statistics in the **DDS_DataWriterProtocolStatus** are not updated correctly. The values that you get after calling the following APIs may be smaller than the actual values:

- DDS_DataWriter::get_datawriter_protocol_status
- DDS_DataWriter::get_matched_subscription_datawriter_protocol_status
- DDS_DataWriter::get_matched_subscription_datawriter_protocol_status_by_locator

[RTI Issue ID CORE-5157]

6.8 Known Issues with TopicQueries

6.8.1 TopicQueries not supported with DataWriters configured to use batching or Durable Writer History

Getting TopicQuery data from a *DataWriter* configured to use batching or Durable Writer History is not supported.

[RTI Issue IDs CORE-7405, CORE-7406]

6.9 Known Issues with C# API

The C# API has the following known issues:

6.9.1 Connext API

- The Distributed Logger API is not yet available in the C# API. However, a custom handler for *Connext* log messages can be used to redirect log messages using the Rti.Config.Logger.MessageLogged event. (RTI Issue ID DISTLOG-122)
- The *FlowController* class is not implemented in the C# API, but FlowControllers can be configured with the Rti.Dds.Core.Policy.Property QoS policy. See DDSPublicationExampleModule_write. (RTI Issue ID CORE-11387)
- The *TopicQuery* class is not implemented in the C# API. TopicQuery can be enabled on the *DataWriter* with the Rti.Dds.Core.Policy.TopicQueryDispatch QoS policy, but it can't be used on the *DataReader*. (RTI Issue ID CORE-11386)
- The interface to implement custom Content Filters is not supported in the C# API. The builtin filters (SQL and String MATCH) are supported. (RTI Issue ID CORE-11388)

6.9.2 IDL-to-C# Code Generation

- Arrays of sequences cannot be used unless the sequence is aliased using a typedef. Otherwise, the array is ignored. (RTI Issue ID CODEGENII-1504)
- The @external annotation is not supported (it is ignored). (RTI Issue ID CODEGENII-1506)
- The @csharp_mapping annotation, which allows customizing some aspects of the IDL-to-C# mapping, is not supported. (RTI Issue ID CODEGENII-1498, CODEGENII-1499)

6.9.3 API Reference

Most classes and methods in the C# API Reference provide only brief descriptions and refer to their counterpart in the C API for their full description. (RTI Issue ID CORE-11634)

6.10 Known Issues with Transports

6.10.1 AppAck messages cannot be greater than underlying transport message size

Why does an AppAck message increase its size? An AppAck message contains a list of sequence number intervals where each interval represents a set of consecutive sequence numbers that have already been acknowledged. As long as samples are acknowledged in order, the AppAck message will always have a single interval. However, when samples are acknowledged out of order, the number of intervals and the size of the AppAck will increase.

This AppAck increase can also happen when a *DataReader* receives many samples from the same *DataWriter* and acknowledges consecutive samples with different payloads, but is unable to immediately send the AppAck message or receive an AppAck confirmation due to network delays or losses. In such cases, the differing

payloads prevent consecutive intervals from being collapsed, potentially creating so many intervals that the AppAck message exceeds the maximum size allowed by the underlying transport.

The property `dds.data_reader.appack_keep_last_payload` can mitigate this issue by reducing the size of AppAck messages, if the application can tolerate receiving only the last payload included in the acknowledgment. By discarding all earlier payloads, *Connext* can merge consecutive samples—either acknowledged without a payload or with discarded payloads—into a single interval, significantly shrinking the AppAck message.

For more information, see Application Acknowledgment in the *Core Libraries User's Manual*.

[RTI Issue ID CORE-5329]

6.10.2 DataReader cannot persist AppAck messages greater than 32767 bytes

A *DataReader* using durable reader state, whose **acknowledgment_kind** (in the ReliabilityQosPolicy) is set to DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE or DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE, cannot persist an AppAck message greater than 32767 bytes.

To recover from the previous error, the *DataReader* must acknowledge samples until the size of the AppAck message goes below the transport message size threshold.

For more information, see the section "Durable Reader State," in the *Core Libraries User's Manual*.

[RTI Issue ID CORE-5360]

6.10.3 Discovery with Connext Micro fails when shared memory transport enabled

Given a *Connext* application with the shared memory transport enabled, a *Connext* Micro 2.4.x application will fail to discover it. This is due to a bug in *Connext* Micro that prevents a received participant discovery message from being correctly processed. This bug will be fixed in a future release of *Connext* Micro. As a workaround, you can disable the shared memory transport in the *Connext* application and use UDPv4 instead.

[RTI Issue ID EDDY-1615]

6.10.4 Communication may not be reestablished in some IP mobility scenarios

If you have two *Connext* applications in different nodes and they change their IP address at the same time, they may not reestablish communication. This situation may happen in the following scenario:

- The applications see each other only from one single network.
- The IP address change happens at the same time in the network interface cards (NICs) that are in the network that is in common for both applications.
- The IP address change on one of the nodes happens before the arrival of the DDS discovery message propagating the address change from the other side.

[RTI Issue ID CORE-8260]

6.10.5 Corrupted samples may be forwarded through Routing Service when using Zero-Copy transfer over shared memory

When using *Zero Copy transfer over shared memory* together with *RTI Routing Service*, *Routing Service* avoids an additional copy of the data by passing a reference to the sample from the input to the output of a route. If the sample is reused and rewritten by the original application *DataWriter* during the time between when the sample was received on the route input and copied into the route output buffer, the forwarded sample will contain the updated, and now invalid, values for the original sample.

This situation can be avoided in a few different ways, with various tradeoffs.

Use automatic application acknowledgment

Using automatic application acknowledgment (**acknowledgment_mode** = APPLICATION_AUTO_ACKNOWLEDGMENT in the Reliability QoS Policy) between the *Routing Service* input *DataReader* and its matching *DataWriters* will avoid the issue.

When using Zero Copy transfer over shared memory, *DataWriters* must loan samples using the **get_loan** API. Only samples that have been fully acknowledged will be returned by the **get_loan** API. This means that if automatic application acknowledgment is turned on, that only samples that the *Routing Service* has already copied and written to the route output will be available for reuse by the original *DataWriter*, because *Routing Service* does not return the loan on a sample until after it is forwarded to the route outputs.

The drawback to this approach is that it requires RELIABLE Reliability. In addition, application-level acknowledgments are not supported in *Connext Micro*, so this approach will not work if *Connext Micro* is the source of the Zero Copy samples.

Ensure that the number of available samples accounts for Routing Service processing time

Regardless of whether you are using *Routing Service*, it is important when using Zero Copy transfer over shared memory to size your resources so that your application can continue to write at the desired rate while the receiving applications receive and process the samples. If you are using *Routing Service* and cannot, or do not wish to, use automatic application acknowledgments, you must take into account the amount of time it will take to receive and forward a sample when setting **writer_loaned_sample_allocation** in the DATA_WRITER_RESOURCE_LIMITS QoS Policy and managing the samples in your application.

[RTI Issue ID CORE-10050]

6.10.6 Network Capture does not support frames larger than 65535 bytes

Network capture does not support frames larger than 65535 bytes. This limitation affects the TCP transport protocol if the **message_size_max** property is set to a value larger than the default one.

[RTI Issue ID CORE-11083]

6.10.7 Shared memory transport in QNX 7.0 and earlier can result in priority inversion

When using QNX 7.0 and earlier with *Connext*, the shared memory transport uses a kind of mutex that does not support priority inheritance. For this reason, using the shared memory transport in QNX can result in priority inversion. Starting with QNX 7.1, priority inheritance is applied to the mutex for the shared memory transport, as described in the fix for CORE-13745 made in a previous release (described here on https://community.rti.com/documentation).

[RTI Issue ID CORE-13745]

6.10.8 Ungracefully terminated QNX processes using SHMEM transport prevents startup of new processes due to unclosed POSIX semaphores (QNX 7.0 and earlier)

If a QNX 7.0 or earlier application using the shared-memory transport was ungracefully shut down, crashed, or otherwise had an abnormal termination while holding a POSIX semaphore used by the transport (for example, while sending data through the shared-memory transport), *Connext* applications launched after that point on the same domain may wait forever for that semaphore to be released.

Workaround for QNX 7.0 and earlier: to enable new applications to start, RTI recommends stopping all applications, then cleaning up the Inter-Process Communication (IPC) resources before starting new applications.

This problem is resolved for QNX 7.1, as described in the fix for CORE-9434 made in a previous release (described here on https://community.rti.com/documentation).

[RTI Issue ID CORE-9434]

6.11 Known Issues with FlatData

6.11.1 FlatData language bindings do not support automatic initialization of arrays of primitive values to non-zero default values

RTI FlatData™ language bindings do not support the automatic initialization of arrays of primitive values to non-zero default values, unless the primitive is an enumeration. It is possible to declare an alias to a primitive member with a default value using the @default annotation, and then to declare an array of that alias. For example:

```
@default(10)
typedef int32 myLongAlias;

struct MyType {
    myLongAlias myLongArray[25];
};
```

The default values of each member of the array in this case should be 10, but in FlatData they will all be set to 0.

[RTI Issue ID CORE-9986]

6.11.2 Flat Data: plain_cast on types with 64-bit integers may cause undefined behavior

The function **rti::flat::plain_cast** is allowed on FlatData samples containing int64_t members, but those members are not guaranteed to have an 8-byte alignment (a 4-byte alignment is guaranteed). Memory checkers such as Valgrind may report errors when accessing such members from the pointer returned by **plain_cast**.

[RTI Issue ID CORE-10092]

6.12 Known Issues with Coherent Sets

6.12.1 Some coherent sets may be lost or reported as incomplete with batching configurations

If *Connext* 6.1.0 receives coherent sets from *Connext* 6.0.0 or lower using batching, coherent sets that are fully received and complete may be lost or marked as incomplete. (If the QoS **subscriber_qos.presentation.drop_incomplete_coherent_set** is set to FALSE, then the samples marked as incomplete won't be dropped.)

[RTI Issue ID CORE-9691]

6.12.2 Copy of SampleInfo::coherent_set_info field is not supported

SampleInfo::coherent_set_info is not available when using take/read operations that do not loan the samples. The **SampleInfo::coherent_set_info** is always set to NULL when you call the take/read operations that do not loan the samples. To get the **coherent_set_info** value, make sure you use the read/take operations that loan the data.

In addition, the copy constructor and assignment operator in the Traditional C++ and Modern C++ APIs do not copy the **SampleInfo::coherent_set_info** field. It is always set to NULL. It is your responsibility to make the copy and handle memory allocation and deletion for this field.

[RTI Issue ID CORE-11215]

6.12.3 Other known issues with coherent sets

Coherent sets are not propagated through *RTI Routing Service* [RTI Issue ID ROUTING-657].

Group coherent sets are not persisted by *RTI Persistence Service* [RTI Issue ID PERSISTENCE-191].

Group coherent sets cannot be stored or replayed with *RTI Recording Service* [RTI Issue ID RECORD-1083].

6.13 Known Issues with Dynamic Data

6.13.1 Conversion of data by member-access primitives limited when converting to types that are not supported on all platforms

The conversion of data by member-access primitives (**get_X()** operations) is limited when converting to types that are not supported on all platforms. This limitation applies when converting to a 64-bit int64 type (**get_longlong()** and **get_ulonglong()** operations) and a 128-bit long double type (**get_longdouble()**). These methods will always work for data members that are actually of the correct type, but will only support conversion from values that are stored as smaller types on a subset of platforms. Conversion to 64-bit int64s from a 32-bit or smaller integer type is supported on all Windows and Linux architectures, and any additional 64-bit architectures. Conversion to 128-bit long doubles from a float or double is not supported.

[RTI Issue ID CORE-2986]

6.13.2 Types that contain bit fields not supported

Types that contain bit fields are not supported by DynamicData. Therefore, when rtiddsspy discovers any type that contains a bit field, *rtiddsspy* will print this message:

```
DDS_DynamicDataTypeSupport_initialize:type not supported (bitfield member)
```

[RTI Issue ID CORE-2977]

6.13.3 Long double not supported for DynamicData in Java API

The Java API does not have DynamicData APIs to handle long double fields. That is, `get_longdouble` and `set_longdouble` don't exist. As a result, it's not possible to work with long double fields of DynamicData samples in the Java API.

[RTI Issue ID CORE-14091]

6.14 Known Issues with RTI Monitoring Library

The following known issues occur in *RTI Monitoring Library*, not in *RTI Monitoring Library 2.0*. See RTI Monitoring Library in the *Connext Core Libraries User's Manual* for more information.

6.14.1 Problems with NDDS_Transport_Support_set_builtin_transport_property() if Participant Sends Monitoring Data

If a *Connext* application uses the **NDDS_Transport_Support_set_builtin_transport_property()** API (instead of the PropertyQosPolicy) to set built-in transport properties, it will not work with *Monitoring Library* if the user participant is used for sending all the monitoring data (the default settings). As a workaround, you can configure *Monitoring Library* to use another participant to publish monitoring data (using the property name **rti.monitor.config.new_participant_domain_id** in the PropertyQosPolicy).

[RTI Issue ID MONITOR-222]

6.14.2 Participant's CPU and memory statistics are per application

The CPU and memory usage statistics published in the *DomainParticipant* entity statistics topic are per application instead of per *DomainParticipant*.

[RTI Issue ID CORE-7972]

6.14.3 ResourceLimit channel_seq_max_length must not be changed

The default value of **DDS_DomainParticipantResourceLimitsQosPolicy::channel_seq_max_length** can't be modified if a *DomainParticipant* is being monitored. If this QoS value is modified from its default value of 32, *Monitoring library* will fail.

[RTI Issue ID MONITOR-220]

6.15 Known Issues with RTI Monitoring Library 2.0

The following known issues occur in *RTI Monitoring Library 2.0*, not in *RTI Monitoring Library*. See MONITORING QosPolicy (DDS Extension) in the *Connext Core Libraries User's Manual* for information on *Monitoring Library 2.0*.

6.15.1 Slow DDS Entity creation in large applications

Enabling *Monitoring Library 2.0* can significantly slow down the creation of new DDS entities in large applications with thousands of existing entities, especially when the library is configured with a non-trivial `<resource_selection>` expression in the `<telemetry_data>` section.

Non-trivial expressions are those that do not follow the simple forms:

- `//*`
- `//<resource_class>/*`

More complex expressions introduce overhead during entity creation, affecting performance.

[RTI Issue ID MONITOR-572]

6.16 Other Known Issues

6.16.1 Possible Valgrind still-reachable leaks when loading dynamic libraries

If you load any dynamic libraries, you may see "still reachable" memory leaks in "dlopen" and "dlclose". These leaks are a result of a bug in Valgrind (https://bugs.launchpad.net/ubuntu/+source/valgrind/+bug/1160352).

This issue affects the *Core Libraries*, SECURITY PLUGINS *(RTI Security Plugins)*, and *TLS Support*.

[RTI Issue IDs CORE-9941, SEC-1026, and COREPLG-510]

6.16.2 64-bit discriminator values greater than (2^31-1) or smaller than (-2^31) not supported

Unions with a 64-bit integer discriminator type containing discriminator values that cannot fit in a 32-bit value are not supported when using the following language bindings:

- C
- Traditional C++
- Modern C++
- C#
- Java
- Python
- DynamicData (regardless of the language)

They are also not supported with ContentFilteredTopics, regardless of the language binding.

Using label values greater than 32-bit may lead to receiving samples with invalid content or to filtering samples incorrectly.

[RTI Issue ID CORE-11437]

6.16.3 Creating multiple DataReaders for the same Topic under the same Subscriber configured with Group Ordered Access is not supported

Creating multiple *DataReaders* for the same *Topic* under the same *Subscriber* configured with PresentationQosPolicy **access_scope** = GROUP and **ordered_access** = TRUE is not supported. If you try to create a second reader in this situation, it will fail to be created and this error will be printed:

```
ERROR [0x0101E967,0x5C3A43B1,0x99D71EB7:0x80000309{Entity=Su,Domain=0}|CREATE␣
↪DR WITH TOPIC FooTopic|LC:Discovery]PRESPsService_createLocalEndpoint:NOT␣
↪SUPPORTED | Creating more than one reader for the same topic within a␣
↪single subscriber using GROUP presentation and ordered_access=true.
```

Instead, in this situation, you will need to use only one *DataReader*, or you will need to create a new *Subscriber* and *DataReader* in the same *DomainParticipant*.

[RTI Issue ID CORE-12448]

6.16.4 With DISALLOW_TYPE_COERCION and Types containing unbounded members, other vendor DataWriters/DataReaders will not match Connext DataWriters/DataReaders

When **reader_qos.type_consistency.kind** is set to DISALLOW_TYPE_COERCION, a *Connext DataReader/DataWriter* for a type containing unbounded collection members will not match with a *DataReader/DataWriter* from another vendor.

A possible workaround is to use ALLOW_TYPE_COERCION or disable type validation by not sending TypeObject information.

[RTI Issue ID CORE-14367]

6.16.5 find_datawriters/readers APIs when used with XML-Based Application Creation is not supported on systems that use older GCC versions

Due to technical limitations, the *Connext* Modern C++ API does not support the `find_datawriters` and `find_datareaders` APIs when used with *RTI Connext XML-Based Application Creation* on machines running GCC 4.8 or earlier. On these platforms, calling these APIs will throw an exception.

[RTI Issue ID CORE-16301]

Chapter 7

Copyrights and Notices

© 2003-2026 Real-Time Innovations, Inc. All rights reserved. June 2026

Trademarks

RTI, Real-Time Innovations, Connext, Connext Drive, NDDS, the RTI logo, 1RTI and the phrase, "Your Systems. Working as one." are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished solely under and subject to RTI's standard terms and conditions available at https://www.rti.com/terms and in accordance with your License Acknowledgement Certificate (LAC) and Maintenance and Support Certificate (MSC), except to the extent otherwise agreed to in writing by RTI.

Third-Party Software

RTI software may contain independent, third-party software or code that are subject to third-party license terms and conditions, including open source license terms and conditions. Copies of applicable third-party licenses and notices are located at community.rti.com/documentation. IT IS YOUR RESPONSIBILITY TO ENSURE THAT YOUR USE OF THIRD-PARTY SOFTWARE COMPLIES WITH THE CORRESPONDING THIRD-PARTY LICENSE TERMS AND CONDITIONS.

Notices

Deprecations and Removals

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

Any deprecations or removals noted in RTI's documentation 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. RTI's current standard terms and support and maintenance policies are available at https://www.rti.com/terms.

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/