

RTI Connex Core Libraries Release Notes

Version 7.6.0



Your systems.
Working as one.

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 7.6.0	6
3.1	Updated default heartbeat periods for faster repairs of lost samples	6
3.2	DNS tracker now enabled by default to reduce chance of discovery problems when using hostnames in peer lists	7
3.3	Extensible Types compliance mask added to all APIs	7
3.4	Updated code generation for 8-bit integer types	7
3.5	Application no longer fails if platform does not support SOCKET_BUFFER_SIZE_OS_MAX	7
3.6	Improved throughput when disable_positive_acks field in DataReaderProtocol QoS policy set to TRUE	8
3.7	New metadata fields for custom filters in modern C++, Java, and Python	8
3.8	TCP Transport now uses POLL socket monitoring by default on Linux systems, offering significantly better responsiveness	8
3.9	Decreased bandwidth usage when not actively debugging remote systems	9
3.10	New built-in profile makes it easier to monitor Connex applications across geographically separated networks	9
3.11	Third-Party Software Changes	10
4	What's Fixed in 7.6.0	11
4.1	Discovery	11
4.1.1	[Critical] Port collision on one channel of multichannel DataWriter caused discovery to fail even if another channel had no port collision	11
4.2	Serialization and Deserialization	12
4.2.1	[Critical] Unexpected error messages when disposing of an instance while using batching and setting serialize_key_with_dispose to TRUE	12
4.3	Usability	12
4.3.1	[Critical] Locator Reachability DataReader could not receive assertions from more than four DomainParticipants	12

4.3.2	[Minor] DDS RTPS_AUTO_ID_FROM_MAC failed if MAC address had all zeros for first 4 bytes or last 4 bytes	13
4.3.3	[Minor] Inconsistent QoS error when enabling Monitoring Library 2.0 with a default profile (is_default_qos = TRUE)	13
4.4	Transports	13
4.4.1	[Major] TCP Transport did not properly enforce system FD_SETSIZE hard limits for select() socket monitoring	13
4.4.2	[Minor] Possible failure to create DomainParticipant if using multicast	14
4.5	Reliability Protocol and Wire Representation	14
4.5.1	[Critical] Unexpected data loss when using batching and finite reader resource limits	14
4.6	Performance and Scalability	15
4.6.1	[Critical] Slower DDS Entity creation in large applications	15
4.7	Debuggability	15
4.7.1	[Major] Duplicate network traffic captures when creating WaitSets, AsyncWaitSets, or GuardConditions	15
4.8	Content Filters and Query Conditions	15
4.8.1	[Critical] Unexpected data loss when using writer-side content filtering and changing QoS	15
4.8.2	[Critical] Historical sample loss for late-joining DataReaders setting protocol.virtual_guid	16
4.9	TopicQueries	16
4.9.1	[Major] max_samples_per_remote_writer resource limit not honored in some cases when using an unkeyed Topic and TopicQueries	16
4.10	Logging	16
4.10.1	[Major] Unexpected warning during DomainParticipant enablement when locator reachability enabled	16
4.10.2	[Major] Misleading warning message when one non-loopback interface doesn't support multicast	17
4.10.3	[Trivial] Isolated RTIOSapiUtility_strerror error on DomainParticipant creation	17
4.11	Dynamic Data	18
4.11.1	[Minor] Unable to set negative values for int8 union discriminators when using Dynamic Data	18
4.12	APIs (C or Traditional C++)	18
4.12.1	[Minor] Traditional C++ DDS_KeyedString and DDS_KeyedOctets constructors allowed 0 for a string size	18
4.13	APIs (Java)	18
4.13.1	[Critical] Java keyhash calculation error with mutable types in XCDR	18
4.14	APIs (Python)	19
4.14.1	[Major] Enhanced logging support in RTI Connex Python API with output_handler_extended	19
4.15	APIs (Multiple Languages)	19
4.15.1	[Major] The result of cdr_serialized_sample_max_size was too small if the maximum size was greater than 4 GB	19
4.15.2	[Major] Non-Java register_type failed if the maximum size was greater than 2 GB	19
4.15.3	[Minor] Enum label not printed when there was only one enum member	20
4.15.4	[Minor] Inconsistent QoS error when setting builtin discovery channels	20
4.16	XML Configuration	20

4.16.1	[Trivial] Unexpected messages printed in console when there was an error parsing XML application configuration file	20
4.16.2	[Major] Union types loaded from XML did not set ID values correctly when using autoid="hash"	20
4.16.3	[Major] XML namespace parsing error in QoS profiles	20
4.17	Entities	21
4.17.1	[Major] Unable to create DistLogger singleton if remote administration was disabled	21
4.18	Crashes	21
4.18.1	[Critical] Potential stack overflow in Connex applications when parsing XML files with a custom DTD with deeply nested entity references	21
4.18.2	[Critical] Possible crash when accessing information about a remote endpoint at the same time that the endpoint was being removed	21
4.18.3	[Major] Changes in RTI Connex logging behavior for Windows desktop applications	22
4.18.4	[Critical] Modern C++ Distributed Logger may have crashed upon instance finalization	22
4.18.5	[Critical] Segmentation fault when using Network Capture APIs concurrently with DomainParticipantFactory deletion	22
4.18.6	[Critical] Potential crash while printing the log message after an error	22
4.18.7	[Critical] Segmentation fault when a DomainParticipant failed to be created	23
4.18.8	[Critical] Segmentation fault and precondition errors with Monitoring Library 2.0 when Entities were deleted	23
4.18.9	[Critical] Possible crash when metric value changed when Monitoring Library 2.0 was being disabled	23
4.19	Hangs	23
4.19.1	[Critical] Calling ignore_participant during a ParticipantBuiltinTopicData-DataReader on_data_available callback triggered by the deletion of a remote DomainParticipant led to a hang or unexpected error message	23
4.19.2	[Critical] Potential deadlock when calling DDS_DomainParticipantFactory_set_qos and DDS_DomainParticipantFactory_finalize_instance	24
4.19.3	[Critical] Potential deadlock when using Network Capture utility APIs concurrently with DDS calls	24
4.19.4	[Critical] Hang of QNX application using shared memory if one of the shared semaphores was in an unusable state	24
4.19.5	[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	24
4.20	Memory Leaks/Growth	25
4.20.1	[Critical] Spike in memory usage when creating a DataWriter using Durable Writer History to restore samples	25
4.20.2	[Major] RTI DDS Spy showed DDS_DataReader_get_matched_publication_participant_data error before leaking memory	25
4.20.3	[Major] DomainParticipantFactory finalization failed even if all the DomainParticipants were deleted	25
4.20.4	[Minor] Memory leak upon Transport TCP creation failure because of invalid configuration	26
4.21	Data Corruption	26
4.21.1	[Critical] Serialization errors in Windows when publishing remote debugging telemetry with Monitoring Library 2.0	26
4.22	OMG Specification Compliance	26

4.22.1	[Major] Incorrect value of an infinite duration in non-Java APIs	26
4.22.2	[Minor] Consistency between max_samples_per_instance and max_samples was not enforced for unkeyed topics	27
4.23	Vulnerabilities	27
4.23.1	[Critical] Potential heap buffer read overflow in Connex applications when using a malicious license string	27
4.23.2	[Critical] Potential invalid read memory access in Connex applications when subscribing to PublicationBuiltinTopicData	28
4.23.3	[Critical] Potential invalid read memory access in Connex applications during endpoint discovery	29
4.24	Other	29
4.24.1	[Critical] Possible Connex validation error while using a valid license	29
4.24.2	[Critical] Data races on architectures with weak memory models	30
4.24.3	[Major] DomainParticipantFactoryQos could not be retrieved from QosProvider	30
4.24.4	[Major] Incorrect hexadecimal formatting for uint8 and octet	30
4.24.5	[Major] Printing to XML missed quotes for external union members	31
4.24.6	[Major] Printing an IDL or XML enum with a default literal produced incorrect output	31
4.24.7	[Major] Printing an enum default literal printed the ordinal value instead of enumerator name	31
4.24.8	[Major] XML parser rejected external members with ranges excluding zero	32
4.24.9	[Major] Incorrect string representation of union DynamicType with arrays of sequences	33
4.24.10	[Major] Error parsing an XML with an external typedef	34
4.24.11	[Major] Invalid escaping of special characters in default strings for XML and IDL output	34
4.24.12	[Major] DynamicType with default string value incorrectly formatted for XML format	34
4.24.13	[Minor] Multiple attachments of same Condition to a WaitSet	34
4.24.14	[Major] Error when parsing an XML with a struct that has an enum member with a default enumerator	35
4.24.15	[Major] XML string literal interpretation divergence between Core Libraries and Code Generator	35
4.24.16	[Major] XML parser did not parse scientific notation when exponent included an explicit positive sign	35
4.24.17	[Major] Error when parsing an XML with a union that has a member with a default value	36
4.24.18	[Minor] Building a Connex application with Wundef warning category could generate unexpected warnings	36
5	Previous Releases	37
5.1	What's New in 7.5.0	37
5.1.1	Python RPC API now fully productized and supported	37
5.1.2	More efficient endpoint discovery for RPC and Request-Reply	39
5.1.3	Improved DataWriter queue sample memory management eliminates required configuration for unbounded types	39
5.1.4	Enhanced out-of-the-box network performance with updated socket buffer sizes	40
5.1.5	Improved scalability for applications using writer-side content filtering	40
5.1.6	Initial discovery performance improvements in large systems with participants containing many endpoints	41

5.1.7	Eliminated requirement to manually clean up thread resources on platforms that support automatic thread-specific storage (TSS) cleanup	41
5.1.8	Improved flow control of fragmented sample repairs	41
5.1.9	RTPS messages containing a malformed inline Qos now trigger a warning, to avoid deserialization errors downstream	42
5.1.10	Capture periodic DomainParticipant announcements in DISCOVERY logging category to enhance debugging of discovery	42
5.1.11	Print incompatible QoS policies in a discovery snapshot for improved discovery debugging	42
5.1.12	Updated default value for socket_monitoring_kind to more performant WINDOWS_IOCP value on Windows systems	42
5.1.13	Support for new metadata fields in content filter expressions	43
5.1.14	InstanceHandle now provides a default constructor	43
5.1.15	Parameter type change to more convenient/appropriate size_t in pluggable transport's to_string APIs	43
5.1.16	Deprecations and Removals	44
	“delete” method of DynamicDataTypeSupport deprecated	44
	dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size removed (except for Java)	44
5.1.17	Third-Party Software Changes	44
5.2	What's Fixed in 7.5.0	45
5.2.1	Discovery	45
	[Critical] Undefined behavior when serializing using any of the builtin discovery plugins	45
	[Critical] Incorrect locators used on newly created DataWriters or DataReaders after a Locator Reachability event	45
	[Major] DomainParticipants using SPDP2 and fragmentation of the participant configuration messages failed to complete discovery	45
	[Major] DomainParticipants failed to announce that endpoint was not reachable when internal clock was monotonic and external clock was real-time	46
	[Minor] DomainParticipants using LBED unregistered remote DataWriters when a remote participant was deleted instead of deleting them	46
5.2.2	Usability	46
	[Minor] Unnecessary QoS consistency checks for best effort DataWriters	46
5.2.3	Transports	46
	[Critical] Shared Memory communications might not have been recovered if a DataReader crashed or was ungracefully killed	46
5.2.4	Reliability Protocol and Wire Representation	47
	[Critical] Writer-side filtered samples not always marked as acknowledged when application acknowledgment was used	47
5.2.5	Performance and Scalability	47
	[Critical] High CPU usage and delayed events in a DataReader managing many instances of a deleted DataWriter	47
5.2.6	Debuggability	47
	[Major] Incorrect source port for some Real-Time WAN Transport packets recorded with network capture	47
	[Trivial] RTI DDS Spy showed unprintable character in help output	48
5.2.7	Content Filters and Query Conditions	48

	[Minor] Setting parameters on QueryCondition could have led to slightly inefficient refiltering	48
5.2.8	Request-Reply and Remote Procedure Calls	48
	[Minor] Request-reply and RPC readers created without explicitly loading Builtin-QosLib::Pattern.RPC did not set reliability.max_blocking_time to 10 seconds	48
5.2.9	Logging	49
	[Trivial] Incorrect informational log messages when adding or removing remote writers in local readers	49
5.2.10	APIs (C or Traditional C++)	49
	[Minor] Incorrect return code for DomainParticipant::assert_liveliness API on disabled participant	49
5.2.11	APIs (Java)	49
	[Major] Unnecessary memory operations when receiving inherited bounded sequences	49
	[Major] For types with strings, sample size calculation in Java may have overestimated serialized size	50
5.2.12	APIs (C# API)	50
	[Trivial] .NET bundle information not displayed in Launcher	50
5.2.13	APIs (Python)	50
	[Critical] Incompatibility between Python API and other APIs when using int8 type	50
	[Major] Python API didn't support reading TopicQuery samples exclusively	50
	[Minor] Python API didn't provide enumeration to set Real-Time WAN Transport in code	51
5.2.14	APIs (Multiple Languages)	51
	[Minor] Some properties no longer accepted LENGTH_UNLIMITED string as a valid value	51
	[Minor] Request-reply and RPC may have used standard PID instead of selected one	51
5.2.15	XML Configuration	51
	[Minor] Possible failure when loading an XML file containing a long comment	51
5.2.16	Instances	52
	[Major] Failure to register instances of mutable flat data types	52
5.2.17	Crashes	52
	[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	52
	[Critical] Possible crash on MIGInterpreter_parse while receiving data at the same time that entities are removed, on architectures with a weak memory model	53
	[Critical] Race condition in RTI Monitoring Library 2.0 caused a crash	53
	[Critical] Segmentation fault if Application Kind contained more than 255 characters when calling RTI_DL_DistLogger_getInstance	53
	[Critical] Potential crash if creation of Publisher or Subscriber failed while database garbage collection process was running	53
	[Critical] Possible crash when calling any write operation on architectures with a weak memory model	54
	[Critical] Stack overflow when parsing very long SQL filter expressions	54
	[Critical] Possible crash when creating DomainParticipants concurrently	54
	[Critical] Crash when writing data on a ContentFilteredTopic in combination with transient durability and delegated reliability	54
	[Major] Possible crash after memory allocation failures	54

	[Major] Using unbounded or very large types with query conditions in Python API could cause crash	55
	[Major] Segmentation fault if application ran out of memory while generating a log message	55
5.2.18	Hangs	55
	[Critical] Potential deadlock when using TCP Transport on Windows systems if WINDOWS_WAITFORMULTIPLEOBJECTS was selected as the socket monitoring API	55
	[Major] Potential deadlock when using Durable Writer History if there was a failure while creating the connection	55
	[Major] Potential deadlock when using DynamicData in Debug libraries if TypePlugin initialization failed	55
	[Major] Potential issues using rti.asyncio.run or asyncio.run with dds.DataReader.take_async	56
	[Major] RPC failures when serializing the return value or out/inout parameters caused a timeout on client side	56
5.2.19	Memory Leaks/Growth	56
	[Critical] Possible unbounded memory growth on DataReader of keyed type after failing to match with a remote DataWriter	56
	[Critical] Memory leaks and errors when using DynamicDataReaders or FlatDataDataReaders plus DDS-fragmentation and compression or encryption	57
	[Critical] Unbounded memory growth when using DynamicDataReaders with multiple data representations and keyed types	57
	[Critical] Unbounded memory growth when using RECOVER instance state consistency and instances were removed from DataReader queue	57
	[Major] Memory leak when parsing invalid module in XML	58
	[Major] Memory leak when using DynamicData with types containing optional members of string aliases or when cloning such types	58
	[Minor] DynamicDataTypeSupport objects not reclaimed by garbage collector	59
	[Minor] Potential DynamicData sample leaked on DataReader deletion	59
5.2.20	Data Corruption	59
	[Critical] Potential corruption in samples with sequences of strings when using Traditional C++ API and std::string	59
	[Critical] Rare race condition may have led to undefined behavior	59
	[Major] Possible data corruption when using ISerializer<T> to serialize a sample	59
5.2.21	Entities	60
	[Major] Error when setting QoS on Publisher or Subscriber without endpoints	60
5.2.22	Interoperability	60
	[Critical] XCDR serialization of a mutable union with a discriminator value that does not select a union member was not compliant with OMG specification	60
	[Critical] RTPS messages containing vendor-specific submessages could have been dropped	61
	[Critical] Multiple RTPS messages received in a single datagram caused issues	61
5.2.23	Vulnerabilities	61
	[Critical] Potential stack buffer overflow in Connex applications when parsing an XML type	61
	[Critical] Potential stack buffer write overflow in license-managed Core Libraries when setting RTI_LICENSE_FILE environment variable	62

	[Critical] Buffer write overflow while parsing malicious license file	63
	[Critical] Potential integer overflow in Connex applications on 32-bit systems when parsing XML files with very large number of default attributes or levels of nesting	64
5.2.24	Other	64
	[Critical] Data races on architectures with weak memory models	64
	[Major] Printing an IDL union with an enum discriminator could have produced an incorrect output	65
	[Major] Incorrect CPU assignment when using THREAD_SETTINGS_CPU_RR_ROTATION in receiver_pool	65
	[Trivial] Possible data race when creating DomainParticipants	65
	[Trivial] Printing a type with an annotation using an enum produced an incorrect output	65
5.3	What's New in 7.4.0	65
5.3.1	Set up reliable communications for fragmented data more easily and with better performance, by removing asynchronous publishing requirement	66
5.3.2	Modern C++ RPC API now fully productized and supported	67
5.3.3	Discover relevant QoS more easily through categorizations added to API Reference HTML documentation	68
5.3.4	Easier QoS configuration for Request-Reply and RPC with new built-in QoS profile	69
5.3.5	Write Python applications faster with reactive subscriptions	70
5.3.6	New Ping built-in type helps debug and prototype with the Python API	71
5.3.7	DDS Spy now prints full timestamps (with date and fractions of a second) and epoch timestamps	71
5.3.8	New DynamicData APIs get and set a union's discriminator value	72
5.3.9	Deprecations and Removals	72
	Simplify QoS configuration by deprecating unnecessary or rarely used QoS settings and policies	72
	dds.sample_assignability.accept_unknown_enum_value and dds.sample_assignability.accept_unknown_union_discriminator properties deprecated	73
	dds.participant.use_45d_compatible_alignment_enforcement property deprecated	73
	EXCLUSIVE_AREA QoS policy removed	73
5.4	What's Fixed in 7.4.0	73
5.4.1	Discovery	74
	[Critical] DNS Tracking not supported with participants using SPDP2	74
5.4.2	Serialization and Deserialization	74
	[Major] Samples containing sequences using discontinuous buffers failed to be serialized	74
	[Major] Calculated maximum size incorrect for unions with nested unions whose members were empty structures	74
5.4.3	Usability	75
	[Major] MultiChannel did not work for synchronous DataWriters with large data in best-effort communications	75
5.4.4	Transports	75
	[Major] Potential high CPU usage if network stack was deleted	75
	[Minor] Wrong configuration of transport_priority QoS on TCP Transport	75
5.4.5	Reliability Protocol and Wire Representation	76
	[Critical] Repair sample was not sent if the sample was smaller than either transport's message_size_max	76

	[Critical] Late-joiner DataReader may have stopped receiving samples from DataWriters using a finite durability.writer_depth	76
	[Major] inactivate_nonprogressing_readers did not always inactivate non-progressing readers	77
	[Major] Reliable large data performance issues due to redundant fragment repairs . . .	77
	[Major] DataWriters with finite durability.writer_depth may have sent repairs over unicast instead of multicast	77
5.4.6	Bandwidth Sensitivity	78
	[Minor] Excessive bandwidth consumption with SPDP2 for peers that were added multiple times	78
5.4.7	Debuggability	78
	[Minor] Receiving large sequence numbers caused DataReader to incorrectly report on_sample_lost() with a negative total_count	78
5.4.8	Content Filters and Query Conditions	78
	[Critical] Failure filtering samples on a DataReader when using DynamicData and Zero Copy	78
5.4.9	TopicQueries	79
	[Major] Response to snapshot TopicQuery may have missed samples for instances that passed the TopicQuery filter	79
5.4.10	Request-Reply and Remote Procedure Calls	80
	[Major] Python RPC API didn't support str arguments	80
5.4.11	Logging	80
	[Trivial] Error messages printed if reader responded to heartbeat from remote writer that was partially removed	80
	[Trivial] Failure getting the Logger instance during instance creation overwrote the verbosity levels of all logging categories to random levels	81
5.4.12	Dynamic Data	81
	[Major] Incorrect DynamicData behavior involving empty structures	81
	[Major] DynamicData to and from JSON conversion did not include union discriminators	82
	[Minor] DynamicData::get_member_info API returned incorrect member_id in some cases	82
5.4.13	APIs (Modern C++ API)	83
	[Major] Possible exception thrown due to incorrect DynamicData move constructor or assignment operator	83
	[Major] Modern C++ API was not enabled for x64Vx7SR0630llvm8.0.0.2_rtp	83
	[Minor] find_member_by_id returned INVALID_INDEX in some cases	84
	[Trivial] SampleProcessor not included by rti/rti.hpp header	84
5.4.14	APIs (Java)	84
	[Major] Unnecessary memory operations when receiving bounded sequences	84
5.4.15	APIs (Python)	84
	[Major] Exception when printing or converting a sequence of octets to string	84
	[Major] Incorrect deserialization of nested union with unknown discriminator	85
	[Major] x64Linux Connex Python API not able to use libraries from \$NDDSHOME/lib	85
	[Minor] QosProvider could not be directly created with QosProviderParams	85
5.4.16	APIs (Multiple Languages)	86
	[Major] DynamicData Errors when operating on unions with unknown discriminator values	86
	[Minor] Creating a DomainParticipant with a negative domain ID was incorrectly allowed	86

5.4.17	XML Configuration	86
	[Major] Could not use multiple <types> tags to create user data types	86
	[Minor] Duplicate type names were allowed in an XML file for defining user data types	88
	[Trivial] DDS-prefixed enumerations caused confusion in XSD files	88
5.4.18	Crashes	89
	[Critical] Potential crash when creating new GuardCondition when low on resources	89
	[Critical] Parsing an XML file with a root “types” tag that contained a “name” attribute led to a segmentation fault	89
	[Critical] Parsing of malformed RTPS packages may have led to bus error in some platforms	89
	[Critical] Potential race conditions in construction or deletion of Requesters and Repliers (C, Traditional C++, C#, and Python)	89
	[Critical] Possible crash when creating dds.DynamicData.Topic with a listener	89
	[Critical] Possible crash after a failure to create Monitoring Library 2.0 object	90
	[Critical] Possible segmentation fault while enabling a DataWriter that enables batching	90
	[Critical] Segmentation fault when copying invalid samples and using Zero Copy transfer over shared memory transport	90
	[Critical] Possible crash when calling DDS_DataWriter_get_matched_subscriptions() or DDS_DataReader_get_matched_publications() on architectures with a weak memory model	91
	[Critical] Possible crash when using QosProvider to load XML file with syntax file:/// on Windows systems	91
	[Critical] Crash on DDS_DomainParticipantFactory_finalize_instance() when using XML-Based Application Creation	91
	[Critical] Parsing of malformed RTPS packets may have led to bus error on some platforms	91
	[Critical] Crash if element in initial_peers longer than 72 characters added	92
	[Major] Potential crash upon serialization, deserialization, or program shutdown when running out of memory	92
	[Major] Possible crash when closing a logger device while it was being used	92
	[Major] Crash during custom transport load when running out of memory	92
	[Major] Precondition failure or segmentation fault when nonBasicTypeName is the name of a module	93
5.4.19	Hangs	93
	[Critical] String built-in type unregister_type API blocked associated DomainParticipant	93
	[Critical] Potential deadlock when using SHMEM transport on Linux, macOS, and LynxOS systems and killing one application	94
	[Critical] Potential deadlock when creating entities in parallel that use Zero Copy transfer over shared memory transport	94
	[Major] Running out of memory during DomainParticipant creation caused DomainParticipantFactory finalization to hang	95
5.4.20	Memory Leaks/Growth	95
	[Critical] Parsing XML configuration with large tokens caused uncontrolled resource consumption	95
	[Critical] Potential unbounded memory growth when using DNS tracking	95
	[Critical] Parsing DTD with many nested/recursive entities could cause uncontrolled resource consumption	96
	[Major] Memory leak when parsing malformed XML defined types	96

	[Minor] Possible memory leak when initializing and finalizing the DomainParticipant-Factory concurrently	96
	[Minor] Memory leak when parsing XML QoS profile with unknown base_name	96
	[Minor] Memory leak during DomainParticipant creation when running out of memory trying to create the event thread	97
5.4.21	Data Corruption	97
	[Major] Parsing of malformed DATA_FRAG submessage may have led to unexpected application behavior	97
5.4.22	OMG Specification Compliance	97
	[Major] Not possible to specify how to deserialize a sample with a discriminator that does not select a member or an unknown enum	97
5.4.23	Interoperability	98
	[Critical] Incorrect handling of RTPS messages with submessages from different participants	98
	[Critical] RTPS checksums incorrect for multiple RTPS messages received in a single datagram	99
5.4.24	Vulnerabilities	99
	[Critical] Stack buffer write overflow while parsing malicious license file	99
	[Critical] Stack buffer write overflow while parsing malicious XML types document	99
	[Critical] Stack buffer write overflow while parsing a malicious XML types document	100
5.4.25	Other	101
	[Critical] Data races on architectures with weak memory model	101
	[Major] Missing implementation of RTI_DL_DistLogger_get_version(), RTI_DL_DistLogger_get_api_build_version(), and RTI_DL_DistLogger_get_api_version_string()	101
	[Major] Incorrect update of DataWriter's pushed_fragment_count protocol statistic under some conditions	101
	[Minor] IDL annotations printed after typedef specifier	102

6	Known Issues	103
6.1	Known Issues with Discovery (SPDP2)	103
6.1.1	Features under future consideration for SPDP2	103
6.1.2	Participants using SPDP2 and allow_unauthenticated_participants fail to communicate if only one participant fails authentication	104
6.2	Known Issues with Serialization and Deserialization	104
6.2.1	Some parameters cannot be received multiple times within same SPDP sample	104
6.2.2	Connnext not compliant with Extended CDR encoding version 2 for types containing arrays and sequences of non-primitive types	105
6.3	Known Issues with Usability	105
6.3.1	Cannot open USER_QOS_PROFILES.xml in rti_workspace/examples from Visual Studio	105
6.3.2	DataWriter's Listener callback on_application_acknowledgment() not triggered by late-joining DataReaders	105
6.3.3	HighThroughput and AutoTuning built-in QoS Profiles may cause communication failure when writing small samples	106
6.3.4	Memory leak if Foo::initialize() called twice	106
6.3.5	Wrong error code after timeout on write() from Asynchronous Publisher	106
6.3.6	Type Consistency enforcement disabled for structs with more than 10000 members	106

6.3.7	Escaping special characters in regular/filter expressions not supported in some cases	107
6.3.8	Non-English Unicode (UTF-8) file names and paths not supported	107
6.4	Known Issues with Code Generation	107
6.4.1	Examples and generated code for Visual Studio 2017 and later may not compile (Error MSB8036)	107
6.5	Known Issues with Instance Lifecycle	108
6.5.1	RECOVER_INSTANCE_STATE_CONSISTENCY setting not fully supported by RTI Infrastructure Services	108
6.5.2	Persistence Service DataReaders ignore serialized key propagated with dispose updates	108
6.5.3	instance_state_consistency_kind QoS cannot be modified before containing entity is enabled	108
6.6	Known Issues with Reliability	109
6.6.1	DataReaders with different reliability kinds under Subscriber with GROUP_PRES- SENTATION_QOS may cause communication failure	109
6.7	Known Issues with Content Filters and Query Conditions	109
6.7.1	Writer-side filtering may cause missed deadline	109
6.7.2	filter_sample_* statistics in DDS_DataWriterProtocolStatus not updated correctly	109
6.8	Known Issues with TopicQueries	109
6.8.1	TopicQueries not supported with DataWriters configured to use batching or Durable Writer History	109
6.9	Known Issues with C# API	110
6.9.1	Connex API	110
6.9.2	IDL-to-C# Code Generation	110
6.9.3	API Reference	110
6.10	Known Issues with Transports	110
6.10.1	AppAck messages cannot be greater than underlying transport message size	110
6.10.2	DataReader cannot persist AppAck messages greater than 32767 bytes	111
6.10.3	Discovery with Connex Micro fails when shared memory transport enabled	111
6.10.4	Communication may not be reestablished in some IP mobility scenarios	112
6.10.5	Corrupted samples may be forwarded through Routing Service when using Zero-Copy transfer over shared memory	112
	Use automatic application acknowledgment	112
	Ensure that the number of available samples accounts for Routing Service processing time	113
6.10.6	Network Capture does not support frames larger than 65535 bytes	113
6.10.7	Shared memory transport in QNX 7.0 and earlier can result in priority inversion	113
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)	113
6.11	Known Issues with FlatData	114
6.11.1	FlatData language bindings do not support automatic initialization of arrays of prim- itive values to non-zero default values	114
6.11.2	Flat Data: plain_cast on types with 64-bit integers may cause undefined behavior	114
6.12	Known Issues with Coherent Sets	114
6.12.1	Some coherent sets may be lost or reported as incomplete with batching configurations	114
6.12.2	Copy of SampleInfo::coherent_set_info field is not supported	115
6.12.3	Other known issues with coherent sets	115
6.13	Known Issues with Dynamic Data	115

6.13.1	Conversion of data by member-access primitives limited when converting to types that are not supported on all platforms	115
6.13.2	Types that contain bit fields not supported	115
6.13.3	Long double not supported for DynamicData in Java API	116
6.14	Known Issues with RTI Monitoring Library	116
6.14.1	Problems with NDDS_Transport_Support_set_builtin_transport_property() if Participant Sends Monitoring Data	116
6.14.2	Participant's CPU and memory statistics are per application	116
6.14.3	ResourceLimit channel_seq_max_length must not be changed	116
6.15	Known Issues with RTI Monitoring Library 2.0	117
6.15.1	Slow DDS Entity creation in large applications	117
6.16	Other Known Issues	117
6.16.1	Possible Valgrind still-reachable leaks when loading dynamic libraries	117
6.16.2	64-bit discriminator values greater than $(2^{31}-1)$ or smaller than (-2^{31}) not supported	117
6.16.3	Creating multiple DataReaders for the same Topic under the same Subscriber configured with Group Ordered Access is not supported	118
6.16.4	With DISALLOW_TYPE_COERCION and Types containing unbounded members, other vendor DataWriters/DataReaders will not match Connex DataWriters/DataReaders	118

7 Copyrights and Notices

Chapter 1

System Requirements

Connex 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 *Connex* 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 *Connex* application for any architecture. You will also need a target installation, which provides the libraries required to build a *Connex* application for that particular target architecture.

Supported platforms, for all products in the *Connex* 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 *Connex* libraries.

Note: For full details on these platforms, including what languages, features, and products they support, see the [RTI Connex 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; Ubuntu® 22.04 LTS, 24.04 LTS	x64	gcc 8.5.0	x64Linux4gcc8.5.0
		Red Hat Enterprise Linux 8	x64	gcc 8.5.0	x64Linux4gcc8.5.0FACE_GP
	Linux (Arm®)	Ubuntu 22.04 LTS, 24.04 LTS	ARMv8	gcc 8.5.0	armv8Linux4gcc8.5.0
		macOS®	macOS 14, 15 (Darwin 23, 24)	ARM64	clang 16.0
	QNX	QNX Neutrino 8.0	x64	qcc_cxx 12.2.0 (LLVM C++ library)	x64QNX8.0qcc_cxx12.2.0
				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 [Memory Consumption](#) 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

*Connex*t 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 *Connex*t 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 *Connex*t.

Although RTPS 2.1 and higher versions are generally interoperable, there may be specific wire protocol interoperability issues between *Connex*t 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

Connex Release	RTPS Standard Version ^{Page 5, 1}	RTPS Protocol Version ²
<i>Connex</i> 7.1.0 and above	2.5 (partial support)	2.5
<i>Connex</i> 6 and 7.0.0	2.3 (partial support)	2.3
<i>Connex</i> 5.2 and 5.3	2.2	2.1
<i>Connex</i> 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 *Connex* 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 *Connex* 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 *Connex* includes partial support for the [OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3](#) (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 Connex 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 *Connex* announces in messages it puts on the wire

Chapter 3

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® Connex® 7.6.0 is an early access release. See the [Connex Versions and Lifecycle](#) page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connex* suite, see those products' release notes on the [RTI Community Portal](#) or in your installation. Or start with the [RTI Connex 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](#).

3.1 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 Connex Core Libraries User's Manual*, for more information.

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

Connex 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 *Connex* 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 Connex Core Libraries User's Manual*, for more information.

3.3 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 *Connex Core Libraries Extensible Types Guide* for more information.

3.4 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`.

3.5 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.

3.6 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.

3.7 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.

3.8 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.

3.9 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 *Connex* 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.

3.10 New built-in profile makes it easier to monitor Connex 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 *Connex* 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*.

3.11 Third-Party Software Changes

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

Table 3.1: New Third-Party Software

Third-Party Software	Version
BearHttpClient	0.2.8
json-builder	Commit ID ffd849

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

Table 3.2: 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 *Connex* products, see the “3rdPartySoftware” documents in your installation: <NDDSHOME>/doc/manuals/connex_dds_professional/release_notes_3rdparty.

Chapter 4

What's Fixed in 7.6.0

This section describes bugs fixed in *Connex* 7.6.0. These are fixes since 7.5.0, described in *Previous Releases*. *Connex* 7.6.0 is an early access release. See the [Connex Versions and Lifecycle](#) page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connex* 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 Discovery

4.1.1 [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]

4.2 Serialization and Deserialization

4.2.1 [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]

4.3 Usability

4.3.1 [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]

4.3.2 [Minor] DDS_RTSPS_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_RTSPS_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]

4.3.3 [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]

4.4 Transports

4.4.1 [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]

4.4.2 [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]

4.5 Reliability Protocol and Wire Representation

4.5.1 [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]

4.6 Performance and Scalability

4.6.1 [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]

4.7 Debuggability

4.7.1 [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]

4.8 Content Filters and Query Conditions

4.8.1 [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 *Data Writer* whose QoS (or whose *Publisher*'s QoS) is incompatible with the *DataReaders*.
- At some point, the *Data Writer* (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]

4.8.2 [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]

4.9 TopicQueries

4.9.1 [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]

4.10 Logging

4.10.1 [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 *Connex* logged this warning during *DomainParticipant* enablement:

```
COMMENTAnonWriterService_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]

4.10.2 [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 *Connex* 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]

4.10.3 [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]

4.11 Dynamic Data

4.11.1 [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]

4.12 APIs (C or Traditional C++)

4.12.1 [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]

4.13 APIs (Java)

4.13.1 [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]

4.14 APIs (Python)

4.14.1 [Major] Enhanced logging support in RTI Connex Python API with `output_handler_extended`

The RTI Connex Python API's `output_handler` implementation allows redirecting Connex log messages to a callable that receives the message as a plain string. This differs from other RTI Connex 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]

4.15 APIs (Multiple Languages)

4.15.1 [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]

4.15.2 [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 *Connex* 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. *Connex* 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]

4.15.3 [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]

4.15.4 [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]

4.16 XML Configuration

4.16.1 [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]

4.16.2 [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]

4.16.3 [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]

4.17 Entities

4.17.1 [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]

4.18 Crashes

4.18.1 [Critical] Potential stack overflow in Connex 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 *Connex* applications of using the previous version was a stack overflow leading to a crash.

[RTI Issue ID CORE-15661]

4.18.2 [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]

4.18.3 [Major] Changes in RTI Connex 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 Connex* log messages. To avoid losing important messages, *RTI Connex* 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 Connex 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 Connex 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]

4.18.4 [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]

4.18.5 [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]

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

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

[RTI Issue ID CORE-15862]

4.18.7 [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]

4.18.8 [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 *Connex* entities (such as *DataWriters*) were deleted. Additionally, when linking against the *Connex* debug libraries, you may have observed precondition errors in the log output.

[RTI Issue ID MONITOR-734]

4.18.9 [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]

4.19 Hangs

4.19.1 [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]

4.19.2 [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]

4.19.3 [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]

4.19.4 [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 *Connex* application crashed or was killed between the creation and the initialization of a shared semaphore, that shared semaphore was unusable. Other *Connex* 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 *Connex* 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]

4.19.5 [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 `ParticipantBuiltinTopicDataReader` `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]

4.20 Memory Leaks/Growth

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

When creating a *Data Writer* 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]

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

When discovering the deletion of a *Data Writer* or *Data Reader* immediately after discovering the deletion of the *DomainParticipant* associated with the *Data Writer* or *Data Reader*, *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]

4.20.3 [Major] DomainParticipantFactory finalization failed even if all the Domain-Participants 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]

4.20.4 [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]

4.21 Data Corruption

4.21.1 [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 *Connex* system.

This problem only affected Windows platforms and the following components:

- *Routing Service*
- *Persistence Service*
- *Data Writers* using zero copy transfer mode

[RTI Issue ID MONITOR-707]

4.22 OMG Specification Compliance

4.22.1 [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]

4.22.2 [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]

4.23 Vulnerabilities

The following vulnerabilities are fixed in this release.

See also [RTI Connex 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 heap buffer read overflow in Connex 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 *Connex* 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 *Connex* 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]

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

An invalid read memory access in *Connex* 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 *Connex* 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 *Connex* 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]

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

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

User Impact without Security

A vulnerability in *Connex* 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 *Connex* 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]

4.24 Other

4.24.1 [Critical] Possible Connex validation error while using a valid license

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

[RTI Issue ID CORE-15738]

4.24.2 [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 *Connex* components.

[RTI Issue ID CORE-15638]

4.24.3 [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]

4.24.4 [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]

4.24.5 [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]

4.24.6 [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]

4.24.7 [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]

4.24.8 [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]

4.24.9 [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]

4.24.10 [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]

4.24.11 [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]

4.24.12 [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="de-fault String"`.

[RTI Issue ID CORE-15847]

4.24.13 [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]

4.24.14 [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]

4.24.15 [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]

4.24.16 [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]

4.24.17 [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_getAnnotationParameter-Value: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]

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

Building a *Connex* 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]

Chapter 5

Previous Releases

5.1 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® Connex® 7.5.0 is an early access release. See the [Connex Versions and Lifecycle](#) page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connex* suite, see those products' release notes on the [RTI Community Portal](#) or in your installation. Or start with the [RTI Connex 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](#).

5.1.1 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.connexdds.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 *Connex* 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.

5.1.2 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.

5.1.3 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 Connex Core Libraries User's Manual* for a detailed explanation of how *DataWriters* manage the memory for serialized samples.

5.1.4 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-connex-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, *Connex* 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 *Connex* to use the maximum socket buffer size allowed by the operating system. If the system does not define a maximum buffer size, *Connex* 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 *Connex* 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 Connex Core Libraries User's Manual*.

5.1.5 Improved scalability for applications using writer-side content filtering

This release enhances the scalability of *Connex* 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.

5.1.6 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.

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

Whenever a *Connex* API is called by a user thread, some resources are stored in thread-specific storage. In previous *Connex* releases, the only way to clean up these resources was to call 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.)

5.1.8 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 Connex Core Libraries User's Manual* for more information.

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

To improve the debuggability of *Connex* applications, this release introduces improvements to the RTPS Inline Qos parsing. Specifically, *Connex* can now detect the presence of an Inline Qos field that is located too early in the RTPS message. When this situation is detected, *Connex* 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.

5.1.10 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.

5.1.11 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 Connex Core Libraries User's Manual* for more information.

5.1.12 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 Connex Core Libraries User's Manual*.

5.1.13 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 Connex Core Libraries User's Manual*.

5.1.14 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()`.

5.1.15 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.

5.1.16 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.

5.1.17 Third-Party Software Changes

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

Table 5.1: 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 *Connex* products, see the “3rdPartySoftware” documents in your installation: `<NDDSHOME>/doc/manuals/connex_dds_professional/release_notes_3rdparty`.

5.2 What's Fixed in 7.5.0

This section describes bugs fixed in *Connex* 7.5.0. These are fixes since 7.4.0, described in *Previous Releases*.

Connex 7.5.0 is an early access release. See the [Connex Versions and Lifecycle](#) page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connex* 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] 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]

5.2.2 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]

5.2.3 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]

5.2.4 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]

5.2.5 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]

5.2.6 Debuggability

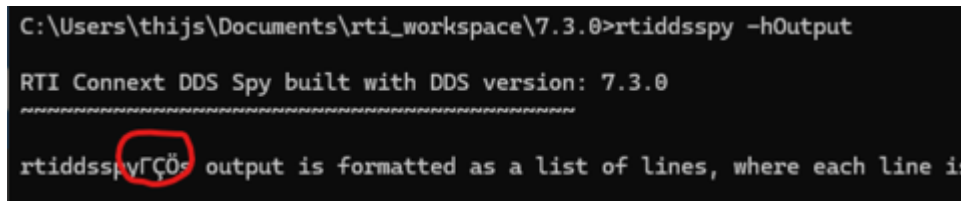
[Major] Incorrect source port for some Real-Time WAN Transport packets recorded with network capture

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:



```
C:\Users\thijs\Documents\rti_workspace\7.3.0>rtiddsspy -hOutput
RTI Connex DDS Spy built with DDS version: 7.3.0
-----
rtiddsspyΓÇÖ output is formatted as a list of lines, where each line is
```

This character has been fixed, and now the help output is printed as expected.

[RTI Issue ID CORE-15093]

5.2.7 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]

5.2.8 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]

5.2.9 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=PREInterParticipantTopic,
↪Type=PREInterParticipantParameter,Domain=0}|LINK 0x0101465D,0x21EAFEE8,
↪0x7AB36BA6:0x000200C2{Type=PREInterParticipantParameter}|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]

5.2.10 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]

5.2.11 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]

5.2.12 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]

5.2.13 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]

5.2.14 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 Connex 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 *Connex* products or connecting *Connex* with other vendors.

[RTI Issue ID REQREPLY-243]

5.2.15 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]

5.2.16 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 *Data Writer'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]

5.2.17 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]

5.2.18 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_WAITFORMULTIPLEOBJECTS` 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]

5.2.19 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 *Connex* 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:RTIOx2022004: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] DynamicDataSupport objects not reclaimed by garbage collector

When a `DynamicDataSupport` was registered with a *DomainParticipant*, the `DynamicDataSupport` 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 `DynamicDataSupport` was deleted before the *DataReader*.

[RTI Issue ID CORE-14324]

5.2.20 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]

5.2.21 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]

5.2.22 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 *Connex*, you must explicitly unset this bit. For instructions, see [Extensible Types Compliance Mask](#) in the *RTI Connex Core Libraries Extensible Types Guide*.

This problem only affected *Connex* 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 *Connex* received RTPS messages containing vendor-specific submessages, *Connex* could have dropped the whole RTPS message mid-processing. This happened when non-*Connex* applications used a vendor-specific submessage whose ID clashed with an ID used by *Connex*.

Now, *Connex* will drop RTPS submessages from non-*Connex* 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 *Connex* 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]

5.2.23 Vulnerabilities

The following vulnerabilities are fixed in this release.

See also [RTI Connex 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 Connex 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 *Connex* 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 *Connex* 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 Connex 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 *Connex* applications of using the previous version varied depending on your *Connex* 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]

5.2.24 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 *Connex* 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 *Connex*. 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]

5.3 What's New in 7.4.0

This section describes what's new in the Core Libraries compared to release [7.3.0](#).

RTI® Connex® 7.4.0 is an early access release. See the [Connex Versions and Lifecycle](#) page for more information on RTI's software release model.

For what's new and fixed in other products in the *Connex* suite, see those products' release notes on the [RTI Community Portal](#) or in your installation. Or start with the [RTI Connex 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](#).

5.3.1 Set up reliable communications for fragmented data more easily and with better performance, by removing asynchronous publishing requirement

In earlier releases of *Connex*, 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, *Connex* 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 *Connex* 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 missing 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 Connex Core Libraries User's Manual*.

5.3.2 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 *Connex* versions.

Note: The RPC Python API continues to be experimental, but will be production-ready in an upcoming release. It incorporates most of these updates, except that it doesn't yet provide the `wait_for_service()` API and doesn't fail when sending a Request before discovering a service. It doesn't provide the option to change the `PID_RELATED_SAMPLE_IDENTITY` parameter, either. (The Request-Reply Python API is production-ready.)

See [Request-Reply Exchanges](#) and [Remote Procedure Calls \(RPC\)](#) in the *RTI Connex Core Libraries User's Manual* for information on these APIs.

5.3.3 Discover relevant QoS more easily through categorizations added to API Reference HTML documentation

Connex Qualities of Service (QoS) policies and fields have been grouped into categories.

The screenshot displays the RTI Connex API Reference documentation for QoS categories. The left sidebar shows a navigation tree where 'QoS Categories' is highlighted. The main content area is titled 'QoS Categories' and includes a 'New' badge. Below the title, there is a link to 'Quality of Service (QoS) categories. More...'. The 'Modules' section lists several categories: Basic QoS, DDS Specification QoS, DDS Extension QoS, Functional Categories, QoS Mutability, Other QoS Attributes, and Deprecated QoS. Each category has a brief description. The 'Detailed Description' section is also visible at the bottom.

The categories are also listed with each QoS field's documentation in the API HTML Documentation.

◆ kind() [1/2]

Reliability & dds::core::policy::Reliability::kind (dds::core::policy::ReliabilityKind the_kind)

Sets the reliability kind.

[default] dds::core::policy::ReliabilityKind_def::BEST_EFFORT for dds::sub::DataReader and dds::topic::Topic, dds::core::policy

Categories New

Immutable, Matching, Discovery Metadata, Reliability

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 Connex 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 Connex DDS API Reference > Infrastructure > QoS > QoS Categories > QoS Mutability**
- **Modules > RTI Connex DDS API Reference > Infrastructure > QoS > QoS Categories > Other QoS Attributes**

5.3.4 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/rticomunity/rticonnextdds-examples/blob/master/examples/connex_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,
)

```

5.3.5 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](#).

5.3.6 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.

5.3.7 DDS Spy now prints full timestamps (with date and fractions of a second) and epoch timestamps

In previous releases, `rtiddspy` 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.

5.3.8 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.

5.3.9 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 *Connex*, 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 Connex Core Libraries User's Manual*.

To see the full list of deprecated QoS settings, see the API Reference HTML documentation under **Modules > RTI Connex 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 *Connex* 4.5d or earlier. It should not be used when your system does not include any legacy (earlier than 4.5d) *Connex*-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`).

5.4 What's Fixed in 7.4.0

This section describes bugs fixed in *Connex* 7.4.0. These are fixes since [7.3.0](#).

Connex 7.4.0 is an early access release. See the [Connex 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.

5.4.1 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]

5.4.2 Serialization and Deserialization

[Major] Samples containing sequences using discontinuous buffers failed to be serialized

Sending a sample containing a sequence that used a discontinuous 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]

5.4.3 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]

5.4.4 Transports

[Major] Potential high CPU usage if network stack was deleted

The deletion of the network stack while a *Connex* 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 *Connex* 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]

5.4.5 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]

5.4.6 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]

5.4.7 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]

5.4.8 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]

5.4.9 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]

5.4.10 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]

5.4.11 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 *Connex* 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()`), *Connex* 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 verbosity levels 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 verbosity levels, 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]

5.4.12 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]

5.4.13 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 *Connex* 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]

5.4.14 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]

5.4.15 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 Connex Python API not able to use libraries from `$NDDSHOME/lib`

On x64Linux, applications using the Connex 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]

5.4.16 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]

5.4.17 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>
</dds>
```

```

    </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 *Connex* 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]

5.4.18 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 *Connex* 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]

[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]

5.4.19 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, *Connex* 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 *Data Writer* or *DataReader* belonging to that *DomainParticipant*.
- Another thread is either:
 - trying to create a *Data Writer* 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]

5.4.20 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 *Connex*, 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

Connex allows adding peers based on a hostname instead of an IP address. When the `dns_tracker_polling_period` is set, *Connex* 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]@udp4://hostname, [4]@udp4://hostname, [2]@udp4://127.0.0.1, [3]@udp4://127.0.0.1, [4]@udp4://127.0.0.1}`. If `hostname` resolved to `127.0.0.1`, the *DomainParticipant* would duplicate the entry for `[4]@udp4://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 `RTIIOsapiContext_associateThread` and `RTIIOsapiContextSupport_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]

5.4.21 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]

5.4.22 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]

5.4.23 Interoperability

[Critical] Incorrect handling of RTPS messages with submessages from different participants

When an RTPS message that contained submessages from multiple participants was received, *Connex* incorrectly treated each submessage as though it were from the participant whose GUID prefix was in the RTPS header. *Connex* 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]

5.4.24 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 *Connex* 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 *Connex* 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]

5.4.25 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 *Connex*.

[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

Connex determines the need for fragmenting a sample using the configured `message_size_max` across all installed transports. If at least one transport requires fragmentation, *Connex* 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), *Connex* 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 typedef 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 `typedef` 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 (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 Connex Core Libraries User's Manual](#) for more information.

6.1.1 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; see [this Knowledge Base article on the RTI Community Forum](#).

- 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]

6.1.2 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]

6.2 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 Connex*, 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\\Documents\rti_workspace\5.3.0\examples\
↳connext_dds\c\

```

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\\Documents\rti_workspace\5.3.0\examples\connext_dds\c\

```

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 $30,720/\text{max_samples}$ 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 Connex 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

Connex 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/connex_dds/c/  
↵hello_UTF/???.xml '
```

This issue also affects files that *Connex* 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 *Connex* 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 environment 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_availability_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 Connex API

- The Distributed Logger API is not yet available in the C# API. However, a custom handler for *Connex* 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

A *DataReader* with `acknowledgment_kind` (in the ReliabilityQoSPolicy) set to `DDS_APPLICATION_AUTO_ACKNOWLEDGMENT_MODE` or `DDS_APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE` cannot send AppAck messages greater than the underlying transport message size.

If a *DataReader* tries to send an AppAck message greater than the transport message size, *Connex* will print the following error message:

```
COMMENDFacade_sendAppAck:!add APP_ACK to MIG  
COMMENDSrReaderService_sendAppAck:!send APP_ACK  
PRESPsService_onReaderAppAckSendEvent:!send acknowledgment
```

To recover from the above error, the *DataReader* must acknowledge samples until the size of the AppAck message goes below the transport message size threshold.

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 been already 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.

For more information, see the “Application Acknowledgment” section 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 Connex Micro fails when shared memory transport enabled

Given a *Connex* application with the shared memory transport enabled, a *Connex* Micro 2.4.x application will fail to discover it. This is due to a bug in *Connex* Micro that prevents a received participant discovery message from being correctly processed. This bug will be fixed in a future release of *Connex* Micro. As a workaround, you can disable the shared memory transport in the *Connex* 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 *Connex* 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 *Data Writer* 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 *Data Writers* will avoid the issue.

When using Zero Copy transfer over shared memory, *Data Writers* 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 *Data Writer*, 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 *Connex Micro*, so this approach will not work if *Connex 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 *Connex*, 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](https://community.rti.com/documentation) 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), *Connex* 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](https://community.rti.com/documentation) 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 *Connex* 6.1.0 receives coherent sets from *Connex* 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 `rtiddspy` discovers any type that contains a bit field, `rtiddspy` will print this message:

```
DDS_DynamicDataSupport_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 *Connex 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 *Connex* 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 *Connex 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 `Presentation-QosPolicy 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 Connex `DataWriters/DataReaders`

When `reader_qos.type_consistency.kind` is set to `DISALLOW_TYPE_COERCION`, a *Connex 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 Type-Object information.

[RTI Issue ID CORE-14367]

Chapter 7

Copyrights and Notices

© 2003-2025 Real-Time Innovations, Inc. All rights reserved. September 2025

Trademarks

RTI, Real-Time Innovations, Connex, Connex 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

Early Access Software

“Real-Time Innovations, Inc. (“RTI”) licenses this Early Access release software (“Software”) to you subject to your agreement to all of the following conditions:

- (1) you may reproduce and execute the Software only for your internal business purposes, solely with other RTI software licensed to you by RTI under applicable agreements by and between you and RTI, and solely in a non-production environment;
- (2) you acknowledge that the Software has not gone through all of RTI’s standard commercial testing, and is not maintained by RTI’s support team;
- (3) the Software is provided to you on an “AS IS” basis, and RTI disclaims, to the maximum extent permitted by applicable law, all express and implied representations, warranties and guarantees, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, satisfactory quality, and non-infringement of third party rights;
- (4) any such suggestions or ideas you provide regarding the Software (collectively , “Feedback”), may be used and exploited in any and every way by RTI (including without limitation, by granting sublicenses), on a non-exclusive, perpetual, irrevocable, transferable, and worldwide basis, without any compensation, without any obligation to report on such use, and without any other restriction or obligation to you; and
- (5) TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL RTI BE LIABLE TO YOU FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, EXEMPLARY OR PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR FOR LOST PROFITS, LOST DATA, LOST REPUTATION, OR COST OF COVER, REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING WITHOUT LIMITATION, NEGLIGENCE), STRICT PRODUCT LIABILITY OR OTHERWISE, WHETHER ARISING OUT OF OR RELATING TO THE USE OR INABILITY TO USE THE SOFTWARE, EVEN IF RTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.”

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/>