RTI Code Generator

Release Notes

Version 3.1.1

Printed in U.S.A. First printing.
March 2022.

Trademarks

Copy and Use Restrictions

Notice

Any deprecations noted in this document serve as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220.

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/

Contents

Chapter 1 Supported Platforms **1**

Chapter 2 Compatibility **2**

Chapter 3 What's New in 3.1.1

3.1 New and Removed Platforms 3

3.2 New C# Language Binding Introduced in 6.1.0 Now Used by Default 3

3.3 Third-Party Software Upgrades 3

Chapter 4 What's Fixed in 3.1.1

4.1 C, Traditional C++, and Modern C++ Generated Code 5

4.1.1 Using an aliased base keyed type caused compilation errors in C and Traditional C++ 5

4.1.2 Compilation error with code generated from a union whose discriminator is an enumerator, when using -namespace and -qualifiedEnumerator options 5

4.1.3 Modern C++ publisher example crashed when IDL type's size was too large for stack variable 6

4.1.4 String elements not allocated to their maximum size in C and traditional C++ for sequences of bounded strings under certain conditions 6

4.1.5 Incorrectly generated code for C/C++ if forward declarations used in sequences of structs and flag -typeSequenceSuffix used 7

4.1.6 Generated code for enum-based union that exhausted all enumerators did not compile in modern C++ 8

4.1.7 Possible memory leak in create_data_w_params if C++ member contained a pointer 8

4.1.8 Possible memory leak in create_data_ex if C++ member contained a pointer 8

4.1.9 Invalid code generated for unions in traditional C++ 9

4.1.10 Float constants defined with exponential were not generated with suffix 'f' in traditional C++ and modern C++ 9

4.1.11 Serialization of string members did not check for null-terminated strings in C, traditional C++, and modern C++ 9

4.1.12 Malformed samples with invalid strings not dropped by DataReader in C, traditional C++, and modern C++ 9

4.1.13 Invalid serialization of samples with types containing nested structures with primitive members that require padding 10
4.2 .NET Generated Code 11
4.2.1 Type code for aliases in C# not generated correctly 11
4.2.2 Incorrectly generated code for C++/CLI if forward declarations used in sequences of structs 11
4.2.3 Generating code for C# API threw null pointer exception for universal platform 11
4.2.4 Incorrect generated code for float constants defined with exponential in C# 11
4.2.5 Generated code for long double constants did not compile in C# 11
4.2.6 Copy constructors for generated classes didn't deep copy sequence members in C# 12
4.3 Generated Code (Multiple Languages) 12
4.3.1 Failure when typedef of enum defined inside a module used @default annotation 12
4.3.2 Code Generator could not parse a file preproccesed with GCC 11 13
4.3.3 CL preproccesor may have caused a deadlock in Code Generator 13
4.3.4 Compilation error with code generated for Java, modern C++, or C# when assigning an enumerator into a constant 13
4.3.5 Float and double ranges may not have been enforced correctly 14
4.3.6 Invalid key deserialization for mutable derived types with key members 14
4.3.7 Deserialization of tampered/corrupted samples may have unexpectedly succeeded 15
4.3.8 Failure when using any @copy directives inside an enum 15
4.3.9 -typeSequenceSuffix was removed from documentation when it should not have been 16
4.3.10 DDS_Int8 incorrectly mapped to unsigned value on QNX systems (on PowerPC and Arm CPUs) and on INTEGRITY systems (on P4080 CPUs) 16
4.4 Fixes Related to Vulnerabilities 16
4.4.1 Arbitrary code execution in Code Generator upon loading malicious template due to vulnerabilities in Velocity 16

Chapter 5 Previous Release

5.1 What's New in 3.1.0 18
5.1.1 New and Removed Platforms 18
5.1.2 New Option to Improve Code Generator Execution Time 18
5.1.3 Deprecated/Removed rtiddsgen Options 18
5.1.4 C, Traditional C++, and Modern C++ Changes 19
5.1.5 Conversion to/from XML 20
5.1.6 Annotations 21
5.1.7 Generated Examples 21
5.1.8 Code Generator Server 23
5.1.9 Other Features 23
5.2 What's Fixed in 3.1.0 24

5.2.1 Generated Code (Multiple Languages) 24
5.2.2 Java Generated Code 27
5.2.3 C, Traditional C++ and Modern C++ Generated Code 31
5.2.4 C++/CLI and .Net Generated Code 36
5.2.5 Conversion to/from XML/XSD 38
5.2.6 Annotations 41
5.2.7 Code Generator Server 43
5.2.8 Examples 44
5.2.9 OMG Specification Compliance 44

Chapter 6 Known Issues

6.1 Classes and Types Defined in Some .NET Namespaces Cannot be used to Define User Data Types 46
6.2 Code Generation for Inline Nested Structures, Unions, and Valuetypes not Supported 47
6.3 .NET Code Generation for Multi-Dimensional Arrays of Sequences not Supported 47
6.4 Request and Reply Topics Must be Created with Types Generated by Code Generator—C API Only 47
6.5 To Declare Arrays as Optional in C/C++, They Must be Aliased 48
6.6 Error Generating Code for Type whose Scope Name Contains Module Called "idl" 48
6.7 Examples and Generated Code for Visual Studio 2017 and later may not Compile (Error MSB8036) 48
6.8 Invalid XSD File from an IDL/XML File if Input File Contains a Range Annotation inside a Structure and a typedef of that Structure 49
6.9 Warnings when Compiling Generated Code for Traditional C++ with -O3 flag and IDL Contains FlatData types 50
6.10 Recursive Structures not Supported 50
6.11 Code Generator Server Cannot be Parallelized 51
6.12 Standalone Types not Supported 51
6.13 uint8 and int8 Types not Fully Supported 51
6.14 64-bit Discriminator Values Greater than (2^31-1) or Smaller than (-2^31) Supported only in Java, no Other Languages 51

Chapter 7 Limitations

7.1 XSD Limitation: Struct with Inheritance can't have Member with Same Name as a Member in Parent 53
7.2 Generated Code for Nested Modules in Ada May Not Compile 54
7.3 Mixing Different Versions of Code Generator Server is not Supported 55

Chapter 1 Supported Platforms

You can run *RTI® Code Generator* as a Java application or, for performance reasons, as a native application that invokes Java. See the *RTI Code Generator User's Manual*.

- As a Java application, *Code Generator* is supported on all host platforms listed in the RTI Core Libraries Release Notes by using the script *rtiddsgen*.
- As a native application, *Code Generator* is supported on the following platforms by using the script *rtiddsgen_server*:
  - All Linux® platforms on Intel® x64 CPUs listed in the *RTI Connext DDS Core Libraries Release Notes*.
  - All Windows® platforms listed in the *RTI Connext DDS Core Libraries Release Notes*.
  - Custom supported platforms: RedHawk™ 6.5.

  Note: POSIX®-compliant architectures that end with "FACE_GP" are not supported.

Chapter 2 Compatibility

For backward compatibility information between 6.1.1 and previous releases, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

Code Generator has been tested with OpenJDK JDK 11, which is included in the installation package.

Chapter 3 What's New in 3.1.1

3.1 New and Removed Platforms

See the *RTI Connext DDS Core Libraries What's New* document for a list of new and removed platforms.

3.2 New C# Language Binding Introduced in 6.1.0 Now Used by Default

By default, when you specify **-language C#**, *Code Generator* now generates code for the new C# API that was introduced in release 6.1.0. If you want to generate code for the legacy C# language, which was used in release 6.1.0 and earlier, specify **-language C# -dotnet legacy**.

The legacy C# language was deprecated in 6.1.0 and will be removed in a future release. Release 6.1.1 is the last release where it is available. See the RTI Code Generator User's Manual for more information.

3.3 Third-Party Software Upgrades

The following third-party software changes have occurred in *Code Generator* in this release:

Third-Party Software	Previous Version	Current Version
AdoptOpenJDK® JRE	11.0.7	11.0.13
Apache Velocity™	1.7	2.3
Apache Log4j 2™	1.2.16	2.17.1
Apache Commons Collections™	3.2.1	No longer used by *Code Generator*. Apache Commons Collections 3.2.1 was a dependency for Apache Velocity 1.7. After upgrading Apache Velocity to 2.3, the Apache Commons Collections 3.2.1 dependency is no longer needed. Therefore, it is no longer included with *Connext DDS*.

Third-Party Software	Previous Version	Current Version
Apache Commons Lang™	2.4	2.6
ANTLR	3.4	3.5.2

Some of these changes may fix potential vulnerabilities. See 4.4 Fixes Related to Vulnerabilities on page 16.

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

Chapter 4 What's Fixed in 3.1.1

4.1 C, Traditional C++, and Modern C++ Generated Code

4.1.1 Using an aliased base keyed type caused compilation errors in C and Traditional C++

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

When the base of a type was an alias of a keyed type, the generated code was incorrect for C and Traditional C++.

This error has been fixed. Now the generated code for a type that inherits from an alias of a keyed type is correct.

[RTI Issue ID CODEGENII-837]

4.1.2 Compilation error with code generated from a union whose discriminator is an enumerator, when using -namespace and -qualifiedEnumerator options

The generated code for a union whose discriminator was an enumerator did not compile if the union was defined in a different module than the one where the enumerator was defined. For example:

```
module A {
    module B {
        enum Color {
            GREEN,
            @default_literal
            BLUE,
            YELLOW,
            RED
        };
};

module C {
```

```
    union myUnion switch(A::B::Color) {
    case A::B::Color::BLUE:
        long defaultUnionMember1;
    };
};
```

The generated code failed with the following error because the enumerator was not accessible in module C:

```
testPlugin.cxx:4504:19: error: use of undeclared identifier 'Color_BLUE'; did you mean
'A::B::Color_BLUE'?
    case (Color_BLUE):
         ^~~~~~~~~~
         A::B::Color_BLUE
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1029]

4.1.3 Modern C++ publisher example crashed when IDL type's size was too large for stack variable

Modern C++ publisher examples created data in the stack. If the IDL type was too large, the creation of the sample crashed.

This release mitigates the problem by allocating the types with arrays in the heap.

[RTI Issue ID CODEGENII-1460]

4.1.4 String elements not allocated to their maximum size in C and traditional C++ for sequences of bounded strings under certain conditions

Starting with 5.3.0, using the **-optimization** command-line option with values 1 or 2 (default is 2) replaced the use of superfluous alias types with the equivalent primitive, enum, or aggregated types.

For example, consider the following IDL:

```
typedef long MyLong;

struct MyStruct {
    MyLong m1;
}
```

The C output with **-optimization** 1 or 2 would be as follows:

```
struct MyStruct {
    int m1;
}
```

The C output with **-optimization** 0 would be as follows:

```
struct MyStruct {
    MyLong m1;
```

```
}
```

This change in behavior introduced a problem with sequences with typedef elements that are bounded strings, because they were resolved to DDS_StringSeq, which represents a sequence of unbounded strings.

For example, consider the following IDL:

```
typedef string<128> MyBoundedString;

struct MyStruct {
    sequence<MyBoundedString> m1;
}
```

The C output with **-optimization** 0 would be as follows:

```
struct MyStruct {
    MyBoundedStringSeq m1;
}
```

The C output with **-optimization** 1 or 2 would be as follows:

```
struct MyStruct {
    DDS_StringSeq m1;
}
```

Unfortunately, the semantic of unbounded string sequences is different from the semantic of bounded string sequences. Specifically, calling maximum in an unbounded string sequence allocates a buffer with NULL strings, whereas calling maximum in a bounded string sequence allocates the string elements to their maximum size.

This regression only affected C and traditional C++. It was not a problem in modern C++ because sequences of bounded and unbounded strings are treated as unbounded sequences.

This problem has been resolved. Now sequences with elements that are typedefs of bounded strings are not resolved to DDS_StringSeq when **-optimization** is set to 1 or 2.

[RTI Issue ID CODEGENII-1489]

4.1.5 Incorrectly generated code for C/C++ if forward declarations used in sequences of structs and flag -typeSequenceSuffix used

The code generated for the following IDL did not add the correct suffix when the **-typeSequenceSuffix** flag was used while running *Code Generator*:

```
struct Foo;
typedef sequence <Foo> SequenceTypeAlias;
struct Foo {
    long myLong;
};
```

This release fixes the issue, generating the correct declaration for the forward-declared sequences with the correct suffix when necessary.

[RTI Issue ID CODEGENII-1520]

4.1.6 Generated code for enum-based union that exhausted all enumerators did not compile in modern C++

The generated code for the following IDL did not compile in modern C++ because *Code Generator* could not find a possible discriminator to initialize/reset the default case:

```
enum TestEnum {
    RED,
    BLUE
};
union TestUnionWithEnum switch (TestEnum) {
    case RED:
    case BLUE:
        SimpleType red_green;
    default:
        long x;
};
```

The issue has been fixed. Now the generated code will be able to initialize/reset the default case.

> **Note:** The OMG 'Interface Definition Language' specification, version 4.2 does not allow a default case in an enum-based union that exhausts all enumerators. This is a *Connext DDS*-specific feature.

[RTI Issue ID CODEGENII-1531]

4.1.7 Possible memory leak in create_data_w_params if C++ member contained a pointer

If errors occurred during creation of data with the method **create_data_w_params()**, some of the allocated memory for members mapped as a pointer may not have been released.

This issue has been fixed. Now all allocated memory is released when errors occur during data creation.

[RTI Issue ID CODEGENII-1566]

4.1.8 Possible memory leak in create_data_ex if C++ member contained a pointer

If errors occurred during creation of data with the method **create_data_ex()**, some of the allocated memory for members mapped as a pointer may not have been released.

This issue has been fixed. Now all allocated memory is released when errors occur during data creation.

[RTI Issue ID CODEGENII-1572]

4.1.9 Invalid code generated for unions in traditional C++

The traditional C++ code generated for unions was wrong and did not compile. For example:

```
union t switch(int32) {
    case 1:
        long l;
    case 2:
        short s;
    default:
        float f;
};
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1586]

4.1.10 Float constants defined with exponential were not generated with suffix 'f' in traditional C++ and modern C++

The constant float defined with exponential values, such as `const float f = 4e12`, did not add the suffix 'f' during the declaration, causing warnings.

This issue has been fixed. Now *Code Generator* will add the suffix 'f' when the float is defined as an exponential value.

[RTI Issue ID CODEGENII-1587]

4.1.11 Serialization of string members did not check for null-terminated strings in C, traditional C++, and modern C++

The code executed by a *DataWriter* that serializes string members in a Topic type did not check that the strings were null-terminated. This may have led to undefined behavior, because the serialization code calls **strlen**.

This problem has been fixed. The serialization code now checks for null-terminated strings with the maximum allowed length and reports the following error if the string is not well-formed:

```
RTIXCdrInterpreter_serializeString:StrStruct:member2 serialization error. String length (at
least 6) is larger than maximum 5
```

[RTI Issue ID CORE-11164]

4.1.12 Malformed samples with invalid strings not dropped by DataReader in C, traditional C++, and modern C++

A *DataReader* may have provided the application a malformed sample containing an invalid value (not null-terminated) for a string member. The string member may not have been null-terminated, resulting in undefined behavior if the application tried to access it.

This issue has been addressed. The *DataReader* will fail to deserialize the sample, and the sample will not be provided to the application.

[RTI Issue ID CORE-11203]

4.1.13 Invalid serialization of samples with types containing nested structures with primitive members that require padding

In *Connext DDS* 6.0.1.20 and 6.1.0, the serialization of samples with a type containing one or more levels of nested complex types, where the nested types only had primitive members, may have failed. This means that a *DataReader* may have received an invalid value for a sample. For example:

```
struct MyType2 {
    long m21;
    long m22;
    double m23;
};

struct MyType {
    long m1;
    MyType2 m2;
};
```

This issue only applied when all of these conditions applied:

- You used XCDR1 data representation.
- The top-level type (MyType above) and the nested type containing only primitive members (MyType2 above) were appendable or final.
- There was a padding in the equivalent C/C++ type between the nested type member (m2 above) and the previous member (m1 above). In the above example, there is a 4-byte padding between m1 and m2 in MyType.

This problem affected DynamicData and the generated code for the following languages: C, C++, C++03, and C++11.

For generated code, a potential workaround to this problem was to generate code with a value of 1 or 0 for the **-optimization** parameter, but this may have had performance implications.

This problem has been resolved.

[RTI Issue ID CORE-11604]

4.2 .NET Generated Code

4.2.1 Type code for aliases in C# not generated correctly

The type code generated for aliases in C# corresponded with the code of the resolved variables. This issue has been fixed. Now, the type code generated for aliases corresponds to the alias itself instead of to the resolved type.

[RTI Issue ID CODEGENII-1500]

4.2.2 Incorrectly generated code for C++/CLI if forward declarations used in sequences of structs

The generated code for C++/CLI may not have declared sequences of structs or unions for any forward-declared type in the correct place. That error may have caused a compilation error in the generated code.

This issue has been fixed. Now the declaration of all sequences of structs and unions related to forward-declared types are declared in the correct place and will not cause compilations errors.

[RTI Issue ID CODEGENII-1521]

4.2.3 Generating code for C# API threw null pointer exception for universal platform

The C# language did not support code generation using **-platform universal**. Previously, *rtiddsgen* did not check if the platform was set to universal when called with the C# language. As a result, *rtiddsgen* crashed.

This issue is now fixed. Now when *rtiddsgen* is called with the command-line arguments **-language C# -platform universal**, *rtiddsgen* will print an error message and finish without crashing.

[RTI Issue ID CODEGENII-1539]

4.2.4 Incorrect generated code for float constants defined with exponential in C#

The generated code for a constant float defined as an exponential, such as that in the following example, did not compile:

```
const float FLOAT_CONST = 7.0e5;
```

Now *Code Generator* generates correct code for this example in C#.

[RTI Issue ID CODEGENII-1558]

4.2.5 Generated code for long double constants did not compile in C#

The following IDL code generated wrong code in C#:

```
const long double LDL = 0.01
```

Now *Code Generator* generates correct code for this IDL code in C#.

[RTI Issue ID CODEGENII-1583]

4.2.6 Copy constructors for generated classes didn't deep copy sequence members in C#

Given the following IDL struct:

```
struct Foo {
sequence<Bar> bar_seq;
};
```

The generated C# Foo class provides a "copy constructor" that allows this:

```
var foo = new Foo();
foo.bar_seq.Add(new Bar());
var foo_copy = new Foo(foo);
```

The expected behavior is a deep copy, including of the sequence elements:

```
foo_copy.bar_seq[0].Equals(foo.bar_seq[0]) && foo_copy.bar_seq[0] != foo.bar_seq[0]
```

However, due to a bug in the code generation, the elements were not cloned (i.e., this was true: **foo_copy.bar_seq[0] == foo.bar_seq[0]**).

This problem has been resolved, and now the sequence elements are cloned as well.

[RTI Issue ID CODEGENII-1680]

4.3 Generated Code (Multiple Languages)

4.3.1 Failure when typedef of enum defined inside a module used @default annotation

Code Generator failed when a typedef of enum defined inside a module used the @default annotation. For example, when using the following IDL:

```
module test {
    enum ValueEnum {
        ZERO,
        NINETYNINE,
        HUNDRED
    };

@default(ZERO)
    typedef ValueEnum myEnum;
};
```

Code Generator failed with the following error:

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 3.1.0, please wait ...
WARN com.rti.ndds.nddsgen.antlr.IdlEnumTree error evaluating expression 'Non literal value ZERO
cannot be resolved'
ERROR com.rti.ndds.nddsgen.Main test.idl line 9:4 Annotation: "ZERO" not found in enum
"ValueEnum"
ERROR com.rti.ndds.nddsgen.Main Fail:  java.lang.Exception: The file couldn't be parsed and the
rawTree wasn't generated
INFO com.rti.ndds.nddsgen.Main Done (failures)
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1179]

4.3.2 Code Generator could not parse a file preproccesed with GCC 11

GCC 11 produced unexpected output when used as a preproccesor. This unexpected output caused an error in *Code Generator*.

This problem has been resolved. *Code Generator* will now work correctly with the output generated by GCC 11.

[RTI Issue ID CODEGENII-1508]

4.3.3 CL preproccesor may have caused a deadlock in Code Generator

When *rtiddsgen* used the preprocessor CL in Windows® and the preprocessor launched a warning, *rtiddsgen* entered a deadlock.

This issue has been fixed. Now *rtiddsgen* cannot enter the deadlock.

[RTI Issue ID CODEGENII-1528]

4.3.4 Compilation error with code generated for Java, modern C++, or C# when assigning an enumerator into a constant

The generated code when assigning an enumerator into a constant did not compile in Java, modern C++, or C#. For example:

```
enum TestEnum {
    FIRST=1,
    SECOND=2,
    THIRD=3
};

const int8 constEnumInt8 = SECOND;
```

The generated code failed with the following error because the enumerator could not be assigned directly into the basic type:

```
constEnumInt8.java:14: error: incompatible types: TestEnum cannot be converted to byte
    public static final byte VALUE =  (byte) (TestEnum.SECOND);
```

```
                                                ^

1 error
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1551]

4.3.5 Float and double ranges may not have been enforced correctly

Float and double ranges may not have been enforced correctly. Float and double member values that should not have passed the check ended up passing it.

This issue only occurred under any of the following conditions:

- For "float":
  - In all languages but Java, when @min was set to -3.4E38 for a member, a value smaller than @min passed the check when it should not have.
  - In all languages but Java, when @max was set to 3.4E38 for a member, a value greater than @max passed the check when it should not have.
- For "double":
  - In all languages but Java, when @min was set to -1.7E+308 for a member, a value smaller than @min passed the check when it should not have.
  - In all languages but Java, when @max was set to 1.7E+308 for a member, a value greater than @max passed the check when it should not have.
- For "float" and "double":
  - In all languages but Java, when the member value was set to INFINITY, samples passed the range check when they should not have.
  - In all languages, when the member value was set to NaN, samples passed the range check when they should not have.

This problem has been resolved.

[RTI Issue ID CORE-11358]

4.3.6 Invalid key deserialization for mutable derived types with key members

In 6.1.0 and 6.0.1.x releases, the key deserialization for mutable derived types with key members when the base does not contain keys was invalid. For example:

```
@mutable
struct Base1 {
```

```
    long m1;
};

@mutable
struct Derived1 : Base1 {
    @key long m2;
};
```

This issue affected the following functionality:

- Calling the APIs **DataWriter::get_key_value** and **DataReader::get_key_value** returned an invalid value.
- When **writer_qos.protocol.serialize_key_with_dispose** was set to TRUE (not the default value) and **writer_qos.protocol.disable_inline_keyhash** was set to TRUE (not the default value), the key-hash calculated on the *DataReader* for a dispose sample sent by a *DataWriter* was invalid. This led to a situation in which a disposed instance was not reported as such on the *DataReader* side.

This issue affected all language bindings except Java and the legacy .NET API. It also affected DynamicData.

This problem has been resolved.

[RTI Issue ID CORE-11378]

4.3.7 Deserialization of tampered/corrupted samples may have unexpectedly succeeded

A *DataReader* may not have detected that a truncated sample due to corruption or tampering was invalid. As a result, the application may have received samples with invalid content.

This issue has been resolved. Now, the deserialization of corrupted samples fails, and they are not provided to the application.

[RTI Issue ID CORE-11494]

4.3.8 Failure when using any @copy directives inside an enum

Code Generator failed when using any @copy directives inside an enum. For example, when using the following IDL:

```
enum MyEnum {
    //@copy-java (/** OutOfRange comment */)
    OutOfRange
};

struct Request {
    boolean Active;
};
```

Code Generator failed with the following error:

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 3.1.0, please wait ...
ERROR com.rti.ndds.nddsgen.Main Fail:  java.lang.ClassCastException: class
com.rti.ndds.nddsgen.antlr.IdlDirectiveTree cannot be cast to class
com.rti.ndds.nddsgen.antlr.IdlEnumeratorTree (com.rti.ndds.nddsgen.antlr.IdlDirectiveTree and
com.rti.ndds.nddsgen.antlr.IdlEnumeratorTree are in unnamed module of loader 'app')
INFO com.rti.ndds.nddsgen.Main Done (failures)
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1152]

4.3.9 -typeSequenceSuffix was removed from documentation when it should not have been

The **-typeSequenceSuffix** command-line option was removed from the -help menu and from the *Code Generator User's Manual*, but it should not have been. It is still supported. This option has been returned to the -help menu and to the *RTI Code Generator User's Manual*.

[RTI Issue ID CODEGENII-1513]

4.3.10 DDS_Int8 incorrectly mapped to unsigned value on QNX systems (on PowerPC and Arm CPUs) and on INTEGRITY systems (on P4080 CPUs)

DDS_Int8 was incorrectly mapped to an unsigned value on QNX® systems (only on PowerPC™ and Arm CPUs) and on INTEGRITY® systems (only on P4080 CPUs). This problem has been fixed. DDS_Int8 is now mapped to a signed value on all platforms.

[RTI Issue ID CODEGENII-1639]

4.4 Fixes Related to Vulnerabilities

This release fixes some potential vulnerabilities, described below.

4.4.1 Arbitrary code execution in Code Generator upon loading malicious template due to vulnerabilities in Velocity

Code Generator had a third-party dependency on Apache Velocity version 1.7. That version of Apache Velocity is known to be affected by a number of publicly disclosed vulnerabilities.

These vulnerabilities have been fixed by upgrading to the latest version of Apache Velocity, 2.3. See 3.3 Third-Party Software Upgrades on page 3.

The impact on *Code Generator* of using the previous version was as follows:

- Exploitable through a compromised local file system containing a malicious *Code Generator* template file.
- *Code Generator* could crash or leak sensitive information. An attacker could execute code with *Code Generator* privileges.
- CVSS v3.1 Score: 7.8 HIGH
- CVSS v3.1 Vector: AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

[RTI Issue ID CODEGENII-1651]

Chapter 5 Previous Release

5.1 What's New in 3.1.0

5.1.1 New and Removed Platforms

See the *RTI Connext DDS Core Libraries What's New* document for a list of new and removed platforms.

5.1.2 New Option to Improve Code Generator Execution Time

Code Generator includes a new option to improve the startup and execution time of the Java Virtual Machine, resulting in an improved code generation time.

To enable this improvement, set the RTIDDSGEN_JVM_OPTIMIZATION environment variable to true. To disable this improvement, unset the environment variable. (By default, it is not set.)

For more information, see the "Boosting Performance" chapter of the *Code Generator User's Manual*.

5.1.3 Deprecated/Removed rtiddsgen Options

This section serves as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220.

5.1.3.1 Deprecated -language C++03 option

The *Code Generator* option **-language C++03** has been deprecated and may be removed in a future version. (The "Modern C++ (C++03)" language option and the "Use legacy C+03/11 plugin" check box in *RTI Launcher*, in the Code Generator dialog, have also been removed.)

For the Modern C++ API, use **-language C++11**; for the Traditional C++ API, use **-language C++**.

After the option is removed, the Modern C++ API will require a C++11 compiler (or newer). The Traditional C++ API will continue to support C++98 compilers.

Please contact support@rti.com if you need to continue using the deprecated option.

5.1.3.2 Removed -legacyPlugin option

The *Code Generator* option **-legacyPlugin** has been removed and is not supported in this release. (The "Use legacy C+03/11 plugin" check box in *Launcher*, in the *Code Generator* dialog, has also been removed.)

5.1.3.3 Removed CORBA Compatibility Kit

RTI CORBA Compatibility Kit is not supported in this release.

The following command-line options have also been removed and no longer work:

- -corba [CORBA Client header file]
- -dataReaderSuffix <suffix>
- -dataWriterSuffix <suffix>
- -orb <CORBA ORB>
- -typeSequenceSuffix <suffix>

5.1.4 C, Traditional C++, and Modern C++ Changes

5.1.4.1 Added support for signed and unsigned 8 bits in language bindings for C, Traditional C++, and Modern C++

Code Generator has added support for int8 and uint8 in the language bindings for C, Traditional C++, and Modern C++. However, these two types will still be considered octets (uint_8) for type matching purposes, maintaining compatibility with older *Connext DDS* versions.

5.1.4.2 IDL enums now map to enum class in C++11

Code generated for **-language C++11** now maps IDL enums to C++'s enum class, instead of the previously used **dds::core::safe_enum** type.

5.1.4.3 IDL to C++11: generated types now include setters by rvalue reference

For each non-primitive struct or union member, **rtiddsgen -language C++11** now generates an additional member function that receives an rvalue reference, allowing to move-assign the value.

For example, given the following IDL:

```
struct Foo {
    string<128> my_str;
};
```

This new setter is generated:

```
class Foo {
    // ...
    void my_str(std::string&& value);
    // ...
};
```

Which allows the following code:

```
Foo foo;
foo.my_str("hello"s + " world"s); // moves the resulting string, instead of copying it
```

Before this release it was possible to achieve the same result with the reference getter, but the setter would make a copy:

```
foo.my_str() = "hello"s + " world"s; // moved
foo.my_str("hello"s + " world"s); // moved since 6.1.0; copied in 6.0.1 and earlier
```

Since 6.1.0, however you assign the value, it will be moved if appropriate.

5.1.5 Conversion to/from XML

5.1.5.1 New format for generated XML type files

This release introduces a new tag structure for generated XML type files. Now they will use the <dds> tag instead of the <types> tag. These new XML files will be validated against a new XSD that allows the use of this <dds> tag.

The use of the <dds> tag allows XML files generated from an IDL to be used straightaway as an *XML-Based Application Creation* type file.

Code Generator still allows as an input an XML file with the old format, but when converting from IDL to XML it will always generate the new format.

This is how an XML file looked before:

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<struct name= "foo">
    <member name="myLong" type="int32"/>
</struct>
</types>
```

This is how the XML file looks now:

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<types>
<struct name= "foo">
    <member name="myLong" type="int32"/>
</struct>
</types>
</dds>
```

5.1.6 Annotations

5.1.6.1 Added support for @data_representation annotation

In release 6.0.1, *Code Generator* added support for the @data_representation annotation (which is equivalent to the @allowed_data_representation annotation) when receiving an input IDL, XML, or XSD file. But when generating an IDL, XML, or XSD file in that release (using one of the "-convertTo" options), *Code Generator* still used @allowed_data_representation in the generated file, regardless of the input annotation (@data_representation or @allowed_data_representation). In this release, *Code Generator* also generates @data_representation annotation when converting to IDL, XML or XSD.

> **Note:** the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3, defines @data_representation as the standard; however, you can still use @allowed_data_representation. *Connext DDS* accepts either.

5.1.6.2 Error now reported if FlatData specified for empty structs

FlatData™ language binding does not support empty structures. Therefore, *Code Generator* will now report an error if @language_binding(FLAT_DATA) annotation is specified for an empty structure.

5.1.7 Generated Examples

5.1.7.1 New generated examples

This release of *Code Generator* includes renewed default generated examples for the following languages:

- C++
- C++11
- Java

To generate these new examples, use the **-example** flag as before. For example:

```
rtiddsgen -language C+/C+11/Java -example <arch> <typeFile>.idl
```

This release also includes new advanced generated examples for the following languages:

- C++
- C++11

- Java

To generate these advanced examples, use the **-exampleTemplate advanced** flag. For example:

```
rtiddsgen -language C+/C+11/Java -example <arch> -exampleTemplate advanced <typeFile>.idl
```

The advanced examples show how to use the same basic *Connext DDS* functionality as the default example, but they also show how to specify a QoS policy, and how to use listeners on the *DataWriters* and *DataReaders* to be notified of events.

Note: Advanced example generation is not supported for Android and INTEGRITY platforms.

5.1.7.2 New base profile when certain annotations are used in IDL

In this version of *Code Generator*, if the last top-level type of the IDL contains the @language_binding (FLAT_DATA) or @transfer_mode(SHMEM_REF) annotations, the generated USER_QOS_PROFILES.xml file will have a different base profile. Instead of "BuiltinQosLib::Generic.StrictReliable," which is the default base profile, the generated USER_QOS_PROFILES.xml file will use "BuiltinQosLib::Generic.StrictReliable.LargeData."

5.1.7.3 New -showTemplates and -exampleTemplate options

This release introduces two new command-line options, **-showTemplates** and **-exampleTemplates**. The **-showTemplates** option prints and generates an XML file containing a list of the available example templates in your *Connext DDS* installation, organized per language. When you use the **-exampleTemplate** option, you can specify one of these example templates, which are placed in **$NDDSHOME/resource/app/app_support/rtiddsgen/templates/example/<language>/<exampleTemplateDirectoryName>/**. You may also create your own templates and place them in this directory.

When you use the **-exampleTemplates** option, *Code Generator* will then generate the example you specified instead of the default one. For example:

```
rtiddsgen -language C++11 -example x64Darwin17clang9.0 -exampleTemplate <exampleTemplateName>
foo.idl
```

See the *Code Generator User's Manual* for more information about how to use these options.

5.1.7.4 Generation of VS2019 project files

Code Generator now generates projects compatible with VS2019. These projects use *Connext DDS* libraries compiled with VS2017. See the *RTI Connext DDS Core Libraries Platform Notes* for more information.

5.1.8 Code Generator Server

5.1.8.1 New log-to-file option for Code Generator server

Code Generator server now has a new option available that enables logging to a file. The option is **-n_log-filepath**. Follow it with a path name, and in that path a file named **CodegenServerLog<portNumber>.txt** will be created with detailed log messages containing the lifecycle of the *Code Generator* server.

5.1.9 Other Features

5.1.9.1 Remote Procedure Calls (RPC) - Experimental Feature

Remote Procedure Calls, or RPC, is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space.

RPC interfaces are defined in IDL, for example:

```
exception TooFastError {
};

@final
struct Coordinates {
    int32 x;
    int32 y;
};

@service
interface RobotControl {
    Coordinates walk_to(Coordinates destination, float speed) raises(TooFastError);
    float get_speed();
    attribute string<128> name;
};
```

From this definition, Code Generator generates a client that can be used as follows:

```
Coordinates final_position = robot_client.walk_to(Coordinates(150, 200), 85.0f);
```

And a service skeleton:

```
class RobotControlExample : public RobotControl {
public:
    Coordinates walk_to(const Coordinates& destination, float speed) override
    {
    …
    }
    …
};
```

The client and service each run on a *DomainParticpant* and under the hood, they use the request-reply communication pattern: the *client* uses a *Requester* to send requests and receive replies; the *service* uses a *Replier* to receive the requests and send the replies.

Note: RPC is an experimental feature available only on C++11, for certain platforms. See the Core Libraries Platform Notes for the supported architectures.

For more details, see the new "Remote Procedure Calls (RPC)—Experimental Feature" chapter in the *RTI Connext DDS Core Libraries User's Manual*.

5.1.9.2 New -strict command-line option

The new **-strict** command-line option enforces types to comply with the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3. When this option is used, it will turn some informational (INFO) messages into errors while running *rtiddsgen*, and no code will be generated. See the *Code Generator User's Manual* for more information.

5.1.9.3 Autocomplete did not work in some XML editors because path in USER_QOS_PROFILES.xml file not found

Autocomplete did not work in some XML editors because the path used by the USER_QOS_PROFILES.xml file could not be found.

Previously, the **.xsd** file location was not compliant with the URI scheme, so some XML editors could not find the **.xsd** file to perform autocompletion. This problem has been resolved.

In the generated USER_QOS_PROFILES.xml file, the location of the **.xsd** at the top of the file used to be something like this:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:/Program Files/rti_connext_dds-6.0.1/resource/schema/rti_dds_
qos_profiles.xsd"
version="6.0.1">
```

Now it correctly uses the **file://** prefix, like this:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///C:/Program Files/rti_connext_dds-
6.1.0/resource/schema/rti_dds_qos_profiles.xsd"
version="6.1.0">
```

5.2 What's Fixed in 3.1.0

5.2.1 Generated Code (Multiple Languages)

5.2.1.1 Typedefs used as baseTypes for inheritance were not resolved

Code Generator optimizes the use of typedefs by resolving them to the basic type when used as a type of a member. This optimization is done in C/C++ when the optimization level is > 0 and by default in Java® and .Net.

This optimization rule, however, was not applied when the typedef was used for a base type in inheritance.

For example:

```
valuetype VT4 {
    public int16 vt4_m1;
};
typedef VT4 myVT4;
@appendable
valuetype VT8: myVT4 {
    public int16 vt8_m1;
};
```

For the above example, *Code Generator* generated the following:

```
public class VT8  extends com.rti.ndds.nddsgen.test.valueType.myVT4
```

The issue has been fixed. Now *Code Generator* generates the base type resolved as follows:

```
public class VT8  extends com.rti.ndds.nddsgen.test.valueType.VT4
```

[RTI Issue IDs CODEGENII-1108]

5.2.1.2 Code generated for unions containing only the default case didn't compile in Java and C++

Code generated for unions containing only the default case was incorrect and didn't compile in Java and C++.

For example, the following IDL:

```
union myUnion switch (int32) {
default:
int32 a;
};
```

Produced the following compilation error in Java:

```
myUnion.java:62: error: -> expected
if (!()) {
^
myUnion.java:62: error: ')' expected
if (!()) {
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1211]

5.2.1.3 Error when serializing non-ASCII unbounded strings in .Net and Java

Previously, the .NET and Java APIs allocated 1 byte for each character in unbounded strings. This allocated memory may not have been enough for serializing/deserializing unbounded strings because a UTF-8 character can use up to 4 bytes if it is a non-ASCII character.

Now *Connext DDS* allocates enough space in the buffer for any UTF-8 characters in unbounded strings.

[RTI Issue ID CODEGENII-1304]

5.2.1.4 Code generator failed when the parameter -V <name>[=<value>] was used

If *Code Generator* was called using the -V <name>[=<value>] parameter, such as follows:

```
rtiddsgen -language C++ -V "test=hello" HelloWorld.idl
```

It failed with the following error:

```
ERROR com.rti.ndds.nddsgen.Main Fail:  com.rti.ndds.nddsgen.Main$ArgumentException: Source file
must have .idl, .xml, or .xsd extension
```

The option was correctly parsed if the space between the flag and the value was removed in the call:

```
rtiddsgen -language C++ -V"test=hello" HelloWorld.idl
```

This issue has been fixed. Now the parameter works correctly with or without the space after the -V.

[RTI Issue ID CODEGENII-1405]

5.2.1.5 Invalid serialization of samples with types containing nested structures with primitive members that require padding

In *Code Generator* 3.0.1 and earlier, the serialization of samples with a type containing two or more levels of nested complex types, where the nested types have primitive members that require padding, may have failed. This means that a DataReader may have received an invalid value for a sample.

Example:

```
// Level-2 Nested type
struct Struct1 {
    uint8 m1;
    uint8 m2;
    int32 m3;
};
// Level-1 Nested type
struct Struct2 {
    int32 m1;
    int32 m2;
    uint8 m3;
    uint8 m4;
    Struct1 m5;
};
struct Struct3 {
    Struct2 m1;
};
```

In the above example, Struct2 and Struct1 are nested, and there is padding between Struct1::m2 (1-byte aligned) and Struct1::m3 (4-byte aligned) of 2 bytes.

This issue only applied to nested types that are appendable or final for XCDR1 data representation or final for XCDR2 data representation.

This problem affected the generated code for the following languages: C, C++, C03, and C++11.

For generated code, a potential workaround to this problem was to generate code with a value of 1 or 0 for the **-optimization**, but this may have had performance implications.

This problem has been resolved.

See also: CORE-10820 in the *RTI Connext DDS Core Libraries Release Notes*, under *What's Fixed in 6.1.0*.

[RTI Issue IDs CORE-10820 and CODEGENII-1486]

5.2.2 Java Generated Code

5.2.2.1 Code Generator generated unused classes for typedefs of typedefs

Code Generator generated holder classes for typedefs of typedefs in Java even when resolving them.

In the following example:

```
typedef sequence<char> MyCharSequence;
typedef char MyCharArray[20];

// The types used in the application code
typedef MyCharSequence appSequence1;
typedef MyCharArray appArray1;

struct structTest {
        appSequence1 appSeq1;
        appArray1    appArr1;
};
```

Code Generator correctly resolved the typedefs, resolving appSeq1 to be a MyCharSequence, but it also generated the classes for the holder class and appSequence that were not needed. As a result, your code might not have compiled if it referenced these classes instead of the resolved class.

This problem has been resolved. Now *Code Generator* does not generate extra classes that are not used.

[RTI Issue IDs CODEGENII-1105 and CODEGENII-267]

5.2.2.2 Generated makefile for Java did not work if Code Generator option "-d" was used with certain inputs

If the *Code Generator* option **-d** was used with certain inputs, the generated makefile for Java may not have worked.

For example, if the following command was used to generate code:

```
rtiddsgen -language java -example x64Darwin17clang9.0 -d java foo.idl
```

The generated makefile would be missing the "java" extension for the source files. For example, the generated makefile would look something like this:

```
JAVA_SOURCES = ./foo. ./fooSeq.
```

But it should have looked something like this:

```
JAVA_SOURCES = ./foo.java ./fooSeq.java
```

The **-d** input was removed from the source file name in the Java makefile. As a result, you could not compile with that makefile.

This issue has been fixed.

[RTI Issue ID CODEGENII-738]

5.2.2.3 Generated code will not compile if an alias of another alias is used as the discriminator of a union in Java

The resolution of a type for multiple levels of aliases may have been wrong under some circumstances. This issue caused the generated code for Java to not compile if the discriminator of a union had multiple levels of aliases.

This issue has been fixed by solving the resolution of the type for multiple levels of aliases.

[RTI Issue IDs CODEGENII-846]

5.2.2.4 Java generated code did not compile when restricting the values of a uint64 (unsigned long long) typedef using a range annotation

Generated code for an input file containing a uint64 (unsigned long long) typedef using a range annotation did not compile. For example:

```
module annotationTests {
    module maxAnnotation {
        @max(6)
        typedef uint64 myUnsignedLongLong;

        struct Example {
            @max(24)
            myUnsignedLongLong myCustomMemberUnsignedLongLong1;
            myUnsignedLongLong myCustomMemberUnsignedLongLong2;
        };
    };
};
```

This example failed to compile, with the following error:

```
[javac] ... myUnsignedLongLongTypeCode.java:28: error: cannot find symbol
[javac]         annotation.max_annotation(new AnnotationParameterValue(new BigInteger("6")));
[javac]                                                                ^
```

```
[javac]   symbol:   class BigInteger
[javac]   location: class myUnsignedLongLongTypeCode
[javac] 1 error
```

This problem has been resolved. Now the generated code imports the BigInteger package when a uint64 typedef uses a range annotation.

[RTI Issue ID CODEGENII-1169]

5.2.2.5 Generated code in Java for @default, @min, @max and @range annotations may not have compiled if their input was an expression (when applied to int16 or octet)

When the annotations @default, @min, @max, or @range were applied to either a short (int16) or an octet, the generated code may not have compiled. This problem occurred only when the annotations contained an expression as the input value.

For example, for the following IDL:

```
@default(4*5)
int16 myMemberShort1;
```

The generated code used to contain:

```
(short)4*5
```

In this code, the casting was applied only to the first number; the resulting multiplication produced an incompatible type. Since a short type was expected, compilation failed.

This issue has been fixed. Now the expression is surrounded by parentheses in the generated code, resulting in the correct casting.

[RTI Issue ID CODEGENII-1189]

5.2.2.6 java.lang classes did not use fully qualified names

Because java.lang classes did not use fully qualified names, you could not name your own types with the same name as a java.lang class. For example, compiling this IDL failed with an error:

```
typedef string String;

struct HelloWorld{
    float foo; }
;
```

'String' should have been fully qualified as 'java.lang.String', but instead resulted in a name collision.

This problem has been resolved. You can now use typedef with the name of a java.lang class, such as String, Integer, or Short.

[RTI Issue ID CODEGENII-1215]

5.2.2.7 Unbounded wchar sequences may have failed with XCDR2 in Java

The use of wchar sequences in Java may have allocated less memory than necessary, causing an error during serialization/deserialization when using XCDR2.

This fix solves this memory allocation error during the serialization/deserialization.

[RTI Issue ID CODEGENII-1248]

5.2.2.8 Method get_key_value failed for wstring arrays in Java with XCDR2

An issue setting the protocol to XCDR2 in Java may have caused silent errors calculating the key of a keyed sample with a wstring array when the sample was received. This issue caused the method **get_key_value** to fail retrieving a sample using its handler.

This release fixes the problem with the protocol by setting the protocol to XCDR2 correctly when necessary. The fix includes changes to avoid silent errors in case something goes wrong.

[RTI Issue ID CODEGENII-1249]

5.2.2.9 Generated Java code may have failed to compile with certain type names

Generated code for Java may have failed to compile because of the collisions of type references and function parameters.

For example, for the following IDL:

```
struct simpleStruct {
int16 myLong;
};
union simpleUnion switch (boolean){
case TRUE:
simpleStruct desc;
};
```

The generated header for the **toString** function would be:

```
public String toString(String desc, int indent)
```

And inside, it would contain the line:

```
strBuffer.append(desc.toString("desc ", indent+1));
```

Where it interpreted *desc.toString* as the parameter instead of the user-defined type. This could be fixed by adding *this.* as follows:

```
strBuffer.append(this.desc.toString("desc ", indent+1));
```

This issue has been resolved. Now the generated code adds *this.* in code where this collision might occur.

[RTI Issue ID CODEGENII-1296]

5.2.3 C, Traditional C++ and Modern C++ Generated Code

5.2.3.1 Suffix for constants in C and C++ generated code either missing or wrong for some types

The generated code for uint16 (unsigned short), int32 (long), and uint32 (unsigned long) constants did not have any suffix. Now *Code Generator* adds the following suffixes:

- "U" for uint16 (unsigned short)
- "L" for int32 (long)
- "UL" for uint32 (unsigned long)

The generated suffix for double constant was wrong if it was assigned an integer and signed number. For example, for "const double MAX = -10;", *Code Generator* would generate "-10*ULL*"; now it generates "-10*LL*".

[RTI Issue ID CODEGENII-439]

5.2.3.2 Incorrect generated code when using IDL whose name starts with a number

The generated code for an IDL whose name started with a number was incorrect and did not compile. The generated code contained some ifdef instructions that started with a number, which was not valid because an identifier must start with a letter (or underscore).

This problem has been resolved. Now invalid identifier characters are converted to '_' in the ifdef instruction.

[RTI Issue ID CODEGENII-513]

5.2.3.3 Generated code for WString when using -useStdString flag did not compile

When the **-useStdString** flag was used to generate code and the IDL file contained a WString type, the generated code would fail to compile. The **-useStdString** flag should only affect Strings, but it was also applied to WStrings, generating incorrect code that did not compile.

This problem has been resolved. Now *Code Generator* only applies the **-useStdString** flag to Strings.

[RTI Issue IDs CODEGENII-1047]

5.2.3.4 Constructors generated in traditional C++ when using -constructor were not exception-safe

The constructors generated in traditional C++ did not free any allocated memory in case of an exception.

Now if an exception is reported in the constructor, *Connext DDS* libraries will catch it, free any allocated memory during the construction, and re-report the exception.

[RTI Issue ID CODEGENII-1218]

5.2.3.5 Code generated for typedef of string in traditional C++ when using -useStdString flag did not compile

When an IDL file contained a typedef of a string, and the flag **-useStdString** was used to generate code, the generated code failed to compile in the following scenarios:

- You were using a version prior to *Connext DDS* 6.0.0.
- You were using *Connext DDS* 6.0.0 or 6.0.1, and you set the -optimization flag to 0.

The generated code in these scenarios contained wrong symbols that caused the compilation to fail.

This issue has been fixed.

[RTI Issue ID CODEGENII-1220]

5.2.3.6 Option -enableEscapeChar was not applied to most identifiers

The use of the option **-enableEscapeChar** did not affect most identifiers.

For example, for the IDL:

```
struct _MyStruct {
    int16 _MyType;
};
```

The generated code for C using **-enableEscapeChar** looked like this:

```
typedef struct _MyStruct {
    DDS_Short   MyMember ;
} _MyStruct ;
```

Where the option **-enableEscapeChar** was applied to the member, but not to the struct.

This issue has been fixed. Now *Code Generator* applies the **-enableEscapeChar** option to all identifiers.

[RTI Issue ID CODEGENII-1263]

5.2.3.7 XCDR2 serialization of a sample for a type with an optional primitive member may have been wrong in some cases

The XCDR2 serialization of a sample for a type with the following properties was incorrect:

- The type had a primitive member 'Pn' (it could be external but not optional) following another primitive member 'Pn-1' that was marked as optional

- The required alignment for 'Pn' was less than or equal to the required alignment for 'Pn-1'
- The optional member was set to NULL in the sample

This issue only affected the following language bindings: C, C++ (traditional and modern), and DynamicData in all languages.

It also affected all languages if using ContentFilteredTopics.

For example, the samples for the following types would not have been serialized correctly when the optional member was set to NULL:

```
struct MyType_1 {
    @optional int32 m1; /* alignment for int32 is 4 */
    int32 m2;  /* alignment for int32 is 4 */
};

struct MyType_1 {
    @optional int32 m1; /* alignment for int32 is 4 */
    double m2;  /* alignment for double is 4 */
};

struct MyType_1 {
    @optional int32 m1;  /* alignment for int32 is 4 */
    @external double m2;  /* alignment for double is 4 */
};
```

On x86 platforms, this issue only resulted in interoperability problems with other DDS vendors. For example, a sample serialized with *Connext DDS* would not have been deserialized correctly by other DDS implementations from different vendors.

On other platforms, such as Arm CPUs, this issue led to a bus error when deserializing.

This problem has been fixed.

[RTI Issue IDs CORE-10254 and CODEGENII-1302]

5.2.3.8 Invalid serialization of samples with types containing members of primitive structures that required padding

In release 6.0.0 and 6.0.1, the serialization of samples with a type containing a member of a primitive structure that required padding only at the end may have been wrong. For example:

```
/* Struct1 requires padding at the end. sizeof(Struct1) is 16 */
struct Struct1 {
    double double1;
    float float1;
};
struct Struct2 {
    float float1;
    float float2;
```

```
};
struct Struct3 {
    Struct1 group1;
    Struct2 group2;
};
struct Struct4 {
    Struct3 msg;
};
```

The serialization of the following sample for Struct4 was wrong: {1.0, 2.0, 3.0, 4.0}. Upon reception, the sample would have been deserialized as {1.0, 2.0, 0.0, 3.0}.

This problem affected DynamicData and the generated code for the following languages: C, C++, C++03, and C++11.

This problem only occurred when all of the following conditions were true. We will use the example above to describe the conditions:

- For non-DynamicData, the code was generated with **-optimization** 2.
- For XCDR2 encapsulation, the structure with padding at the end (Struct1) must be @final. For XCDR1 encapsulation, Struct1 must be @appendable or @final.
- The member of Struct1 (group1) was followed by another member (group2) whose type had an alignment less than or equal to the alignment of the last member (float1) of Struct1. In the example above, group1 (where float1 has an alignment of 4) is followed by group2 (whose first member, float1, had an alignment of 4).
- The member of Struct1 must have been on a second or higher level of nestedness.

This problem has been resolved.

[RTI Issue ID CORE-10311 and CODEGENII-1315]

5.2.3.9 Header for internal API type>Plugin_deserialize() was generated without body

The following header was generated without any implementation in the header files for C and Traditional C++ . This is an internal API that has not been needed since *Connext DDS* 6.0.0:

```
RTIBool
<type>Plugin_deserialize(
    PRESTypePluginEndpointData endpoint_data,
    Foo **sample,
    RTIBool * drop_sample,
    struct RTICdrStream *stream,
    RTIBool deserialize_encapsulation,
    RTIBool deserialize_sample,
    void *endpoint_plugin_qos);
```

This issue has been fixed. The method is no longer generated.

[RTI Issue ID CODEGENII-1334]

5.2.3.10 Incorrectly generated code for C/C++ if forward declarations used in sequences of structs

The generated code for C and traditional C++ may not have declared sequences of structs for any forward-declared type in the correct place. That error may have caused a compilation error in the generated code.

This issue has been fixed. Now the declaration of all sequences of structs related to forward-declared types will be declared in the correct place and will not cause compilation errors.

[RTI Issue ID CODEGENII-1428]

5.2.3.11 Incorrectly generated code for C/C++ if union forward declarations used in sequences

Previously, the generated code for C and traditional C++ may not have declared sequence unions in the correct place for any forward-declared type. That issue may have caused compilation errors in the generated code.

This issue has been fixed. Now, the declaration of all the sequence unions related to forward-declared types will be declared in the correct place and will not cause compilations errors.

[RTI Issue ID CODEGENII-1439]

5.2.3.12 Possible serialization error with derived types when using Code Generator flag -virtualDestructor in C++

When the flag **-virtualDestructor** was passed to the *Code Generator* (*rtiddsgen*) to generate traditional C++ code for a type with inheritance, such as:

```
valuetype myValueType {
public int32 l1;
};
valuetype myValueTypeDerived : myValueType {
public int32 l2;
}
```

this may have caused a serialization error or incorrect data to be received when the derived type was transmitted. This release fixes the serialization issue and the data is transmitted correctly.

[RTI Issue ID CODEGENII-1458]

5.2.3.13 Interoperability issue between Java/NET and C/C++/modern C++ applications when using keyed types and XCDRv1 encapsulation

In *Code Generator* 3.0.0 and 3.0.1, the instance keyhash for a keyed type using XCDR (Extensible CDR version 1) encapsulation was calculated differently in the Java, and .NET languages when the code for the keyed type was generated using the **-optimization 0** option and the keyed type contained one key member

whose type was a typedef of a struct/valuetype type in which only some of the members were marked as **@key** fields. For example:

```
struct SimpleKeyedType
{
    @key octet m1;
    octet m2;
};
typedef SimpleKeyedType SimpleKeyedTypeAlias;
struct KeyedType
{
    @key SimpleKeyedTypeAlias m1;
};
```

The right calculation was done in Java.

As a result, the subscribing application might have observed some unexpected behavior related to instances. Specifically, the call to **DataReader::lookup_instance()** might have failed and returned HANDLE NIL even if the instance was received.

This also affected compatibility with the languages C, C++, and Modern C++ in 5.3.1 or earlier releases.

This problem has been resolved.

See also: CORE-11290 in the *RTI Connext DDS Core Libraries Release Notes*, under *What's Fixed in 6.1.0*.

[RTI Issue IDs CORE-11290 and CODEGENII-1485]

5.2.4 C++/CLI and .Net Generated Code

5.2.4.1 IDL file containing a struct with multiple optional enums generated incorrect code for .Net language

The use of multiple optional enums in a struct generated incorrect code for the .Net language. This resulted in a compilation error. This problem has been resolved.

[RTI Issue ID CODEGENII-1125]

5.2.4.2 Multidimensional arrays may have caused serialization error in C++/CLI and .Net

The serialization in C++/CLI and .Net when using XCDR2 for types containing a multidimensional array of primitive types that are doubles may have been incorrect.

As a result, a subscriber application using XCDR2 in any language and receiving samples from an XCDR2 C++/CLI or .Net publisher application might have received incorrect data or failed to deserialize the received sample.

This issue has been fixed.

[RTI Issue ID CODEGENII-1231]

5.2.4.3 Incorrect deserialization in .Net of samples from a keyed mutable type containing an extensible element when published from a writer with disable_inline_keyhash set to true

The deserialization in .Net of a sample of a keyed mutable type containing an extensible element, where the extensible element is smaller in the published type, was incorrect if the sample was published by a *DataWriter*, of any language, that had set the **writer_qos.protocol.disable_inline_keyhash** QoS to true.

An example of this would be the following, where MutableV2Struct is the published type and MutableV1Struct is the subscribed type:

```
module XTypes {
    struct ExtensibleBaseStruct {
        int32 m1; //@key
    }; //@top-level false

    struct ExtensibleStruct: ExtensibleBaseStruct {
        int32 m2;
        int16 m3;
    }; //@top-level false

    struct MutableV1Struct {
        ExtensibleStruct m1;
        string m2; //@key
    }; //@Extensibility MUTABLE_EXTENSIBILITY

    struct MutableV2Struct {
        ExtensibleBaseStruct m1;
        string m2; //@key
    }; //@Extensibility MUTABLE_EXTENSIBILITY
};
```

As a result of this problem, the .Net subscriber might have reported an error like the following and would not have been able to deserialize the received sample:

```
PRESCstReaderCollator_serializedKeyOrSampleToKeyHash:!serialized sample to keyhash
PRESCstReaderCollator_getSampleKeyHashes:!serialized key/sample to keyhash
PRESCstReaderCollator_storeInlineQos:!get sample keyHashes
PRESCstReaderCollator_storeSampleToEntry:!store inline qos in entry
PRESCstReaderCollator_newData:!get entries
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1234]

5.2.4.4 Incorrect deserialization in .Net of samples from a keyed type containing an array/sequence of wstring when published from a writer with disable_inline_keyhash set to true using XCDR2

The deserialization in .Net of a sample of a keyed type containing an array/sequence of wstring, where the array/sequence is placed before the last keyed element in the type, was incorrect if the sample was published by a *DataWriter*, of any language, that had set the **writer_qos.protocol.disable_inline_keyhash** QoS to true and that was using the XCDR2 data representation.

An example of this type would be the following:

```
struct Sequence {
        sequence<wstring<5>,10> myWstringSeq;
        int32 id; //@key
};
```

As a result of this problem, the .Net subscriber might have reported an error like the following and would not have been able to deserialize the received sample:

```
PRESCstReaderCollator_serializedKeyOrSampleToKeyHash:!serialized sample to keyhash
PRESCstReaderCollator_getSampleKeyHashes:!serialized key/sample to keyhash
PRESCstReaderCollator_storeInlineQos:!get sample keyHashes
PRESCstReaderCollator_storeSampleToEntry:!store inline qos in entry
PRESCstReaderCollator_newData:!get entries
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1261]

5.2.4.5 Performance issues creating entities with sequences of complex types

Under some circumstances, bounded sequences of complex types could cause a performance issue during the creation of entities such as a *DataReader* and *DataWriter* in C#. The problem was caused by the calculation of sample sizes.

The algorithm to calculate the sample size in C# has been improved, increasing the performance during the creation of entities.

[RTI Issue ID CODEGENII-1361]

5.2.5 Conversion to/from XML/XSD

5.2.5.1 Code Generator failed if an included XSD file contained an array of any element

When generating code from an XSD that included another XSD file, an error like the following was reported and no code was generated, if the included XSD file contained an array of any element:

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 3.0.1, please wait ...
ERROR com.rti.ndds.nddsgen.Main rpc_types.xsd line 20:124 member type 'dds::EntityId_t_
entityKey_ArrayOfbyte' not found
```

```
ERROR com.rti.ndds.nddsgen.Main Fail: java.lang.Exception: The file couldn't be parsed and the
rawTree wasn't generated
INFO com.rti.ndds.nddsgen.Main Done (failures)
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1204]

5.2.5.2 Failure to process included file in XML/XSD when included file was specified using absolute path

Code Generator failed to process an included file in XML/XSD when the included file was specified using its absolute path. For example:

/home/<username>/debug/codegenii

├── include

│ └── include.xml

└── test

│ └── test.xml

Where the content of each XML is the following:

- include.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="...">
<struct name= "incl">
    <member name="a" type="int32"/>
</struct>
</types>
```

- test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="...">
<include file="/home/<username>/debug/codegenii/include/include.xml"/>
<struct name= "test">
    <member name="A" type="nonBasic"  nonBasicTypeName= "incl"/>
</struct>
</types>
```

Code Generator failed with the following error:

```
<username>@...    ~/debug/codegenii    ~/rti_connext_dds-6.0.1/bin/rtiddsgen -example universal
test/test.xml
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 3.0.1, please wait ...
```

```
WARN com.rti.ndds.nddsgen.xml.XMLHandler Included file
/home/<username>/debug/codegenii/test/home/<username>/debug/codegenii/include/include.xml not
found
WARN com.rti.ndds.nddsgen.xml.XMLParser The included file wasn't parsed
ERROR com.rti.ndds.nddsgen.Main test.xml line 5:63 member type 'incl' not found
ERROR com.rti.ndds.nddsgen.Main Fail:  java.lang.Exception: The file couldn't be parsed and the
rawTree wasn't generated
INFO com.rti.ndds.nddsgen.Main Done (failures)
```

As you can see from the error, *Code Generator* added the path where the input file is located (/home/<username>/debug/codegenii/test/) at the beginning when trying to look for the included file (which is located in /home/<username>/debug/codegenii/include/include.xml).

This problem has been fixed. You can now specify the included file using its absolute path, and *Code Generator* finds the file; you no longer get this error.

[RTI Issue ID CODEGENII-1213]

5.2.5.3 Incorrect generated XML for derived struct with @resolve-name false annotation

When converting an IDL to XML, if the IDL contained a derived struct with the @resolve-name false annotation, the generated XML was incorrect.

For example, consider the following IDL:

```
struct baseStruct {
    int16 a;
};

struct derivedWithoutResolvingName : baseStruct {
    int16 b;
}; //@resolve-name false
```

When using the *Code Generator*-**convertToXml** option on this IDL, the generated XML file contained the following line:

```
<struct name= "DerivedWithoutResolvingName" baseType="simpleBaseStruct resolveName="false">
```

This line is missing the final quotation mark in baseType="simpleBaseStruct.

This issue has been fixed.

[RTI Issue ID CODEGENII-1297]

5.2.5.4 Error generating code from XML for certain structs with FlatData and mutable extensibility

When an XML file contained a struct with FlatData and mutable extensibility, if this struct had non-fixed-size types, code generation would report an error and fail.

For example, for an XML containing:

```
<struct name= "MyStruct" extensibility= "mutable" languageBinding="flat_data">
<member name="MyString" stringMaxLength="128" type="string"/>
</struct>
```

Code Generator would show the following error:

```
ERROR com.rti.ndds.nddsgen.Main flat.xml line 4:63 @language_binding(FLAT_DATA) in a final type
requires fixed-size types; 'MyString' cannot be a string
```

This issue has been resolved. *Code Generator* is now able to generate code for XML types that occur in this scenario.

[RTI Issue ID CODEGENII-1342]

5.2.6 Annotations

5.2.6.1 Annotations @default, @min, @max, and @range for uint32 did not properly validate input value

When the annotation @default, @min, @max, or @range was used with the type uint32, *Code Generator* did not properly check if that value was within the uint32 range.

For example, for the following IDL:

```
struct foo {
@max(4294967295)
uint32 myUnsignedShort;
};
```

Code Generator reported an error and did not generate code. This should not happen, however, since the uint32 range goes up to 4294967295.

This issue has been fixed.

[RTI Issue ID CODEGENII-1137]

5.2.6.2 Code generation failed when there was a @default annotation applied to a typedef that was used within a @range, @min, or @max annotation

When the @default annotation was specified for a typedef, and that typedef was used later with a @range, @min, or @max annotation, code generation might have failed. The @default annotation defined in the typedef was not applied to the variable using it.

For example, for the following IDL:

```
@default(3)
typedef int32 myLong;
struct myStruct {
    @min(2)
    myLong myStructLong;
};
```

Code Generator's code generation incorrectly reported an error saying "@min value (2) is greater than implicit @default value (0)". Although the default value for an int32 (long) is 0, the inherited default value for myStructLong is 3, so there should be no error.

This issue has been fixed. *Code Generator* now uses the @default value inherited from the typedef.

[RTI Issue ID CODEGENII-1138]

5.2.6.3 Generated code for @default(true/false) did not compile

When the @default() annotation was used with the value true/false in lower case, the generated code failed to compile.

This issue has been resolved. Now true/false in lower case are not supported values for the @default annotation; *Code Generator* will report an error message and not generate code.

[RTI Issue ID CODEGENII-1173]

5.2.6.4 Annotations @default, @min, @max, and @range for int64/uint64 did not validate input value

When the annotation @default, @min, @max, or @range was used with the type int64/uint64, *Code Generator* did not check if that value was within the int64/uint64 range.

For example, for the following IDL:

```
struct foo {
    @min(-1)
    uint64 myUnsignedLongLong;
};
```

Code Generator did not report an error; however, it should have reported an error, since uint64 cannot have negative values.

This issue has been fixed.

[RTI Issue ID CODEGENII-1241]

5.2.6.5 Code Generator wrongly allowed a union to be final and FlatData

Code Generator allowed a union to be FlatData and have final extensibility, which should not be allowed.

This issue has been resolved. Now *Code Generator* reports an error if the type contains a FlatData union with final extensibility.

[RTI Issue ID CODEGENII-1345]

5.2.7 Code Generator Server

5.2.7.1 Code Generator server hung on client side waiting for messages from server

In previous *Code Generator* server versions, the client would wait indefinitely for server messages. This could result in a deadlock when there was another application different than the *Code Generator* server listening on that port.

This problem has been resolved. Now the *Code Generator* server sends a handshake message to the client after accepting the connection. In the client, there is now a timeout when waiting for the handshake message. As a result, if there is another application listening on that port that is different than the *Code Generator* server, the *Code Generator* server will time out after a short amount of time if the *Code Generator* client doesn't receive the handshake message.

See Chapter 7 "Boosting Performance with Server Mode" in the *Code Generator User's Manual* for more information.

[RTI Issue ID CODEGENII-1324]

5.2.7.2 Code Generator server mode was run when wrong flag was specified

Code Generator ran in server mode if it received either a hyphen - or an empty string "" as an option.

For example:

```
rtiddsgen -language C -example x64Darwin17clang9.0 test.idl -
```

or

```
rtiddsgen -language C -example x64Darwin17clang9.0 test.idl ""
```

This issue has been fixed. *Code Generator* will now report an error in these scenarios.

[RTI Issue ID CODEGENII-1354]

5.2.7.3 NullPointerException when using Code Generator in server mode without specifying an architecture if server mode was previously run specifying an architecture

Code Generator threw a NullPointerException in server mode when run without specifying an architecture if it was previously run specifying an architecture. Running without specifying an architecture means not using any of the options that configure the architecture used to generate code, such as **-example** and **-create**.

For example, this command worked:

```
~/rti_connext_dds-6.0.1/bin/rtiddsgen_server -example i86Linux3.xgcc4.6.3
~/path/to/file/test.idl
```

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 3.0.1, please wait ...
INFO com.rti.ndds.nddsgen.Main Done
```

This command, when run after the first one, didn't work:

```
~/rti_connext_dds-6.0.1/bin/rtiddsgen_server ~/path/to/file/test.idl
```

```
ERROR com.rti.ndds.nddsgen.Main Fail: java.lang.NullPointerException
INFO com.rti.ndds.nddsgen.Main Done (failures)
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1264]

5.2.8 Examples

5.2.8.1 Autocomplete did not work in some XML editors because the path used by the USER_QOS_PROFILES.xml file could not be found

In the generated USER_QOS_PROFILES.xml file, the location of the .xsd at the top of the file used to be something like this:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:/Program Files/rti_connext_dds-6.0.1/resource/schema/rti_dds_
qos_profiles.xsd"
version="6.0.1">
```

Now it correctly uses the "file://" prefix, like this:

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///C:/Program Files/rti_connext_dds-
6.0.1/resource/schema/rti_dds_qos_profiles.xsd"
version="6.0.1">
```

Previously, the .xsd file location was not compliant with the URI scheme, so some XML editors could not find the .xsd file to perform autocompletion. This problem has been resolved.

[RTI Issue ID CODEGENII-319]

5.2.8.2 -d64 flag missing in generated Java makefile for Red Hat Enterprise Linux 8.0

The generated Java makefile for Red Hat® Enterprise Linux® 8.0 (x64Linux4gcc7.3.0) was missing the flag **-d64**. This flag caused the compiled code to report an error when executed in a 32-bit environment.

This problem has been resolved.

[RTI Issue ID CODEGENII-1332]

5.2.9 OMG Specification Compliance

5.2.9.1 @key fields set in derived structure

Previously, @key fields could be set in a derived structure.

However, the OMG 'Extensible and Dynamic Topic Types for DDS' specification, version 1.3 states the following:

> *"A Structure Type may inherit from another Structure Type as long as the following conditions are met:*
>
> ...
>
> ...
>
> - *The derived type does not define any key fields. This ensures the key fields of the derived type are the same as those of the base root type."*

In this release, *Code Generator* will report an informational (INFO) message for keyed derived structures when running *rtiddsgen*, but still generate code. If you want *Code Generator* to enforce the specification, use the **-strict** option when running *rtiddsgen*. This option will report an error when there are keyed derived structures and not generate code. See the *Code Generator User's Manual* for more information.

[RTI Issue IDs CODEGENII-1116]

5.2.9.2 Non-standard 64-bit integer types generated for C++11

IDL code generated for the Modern C++ API (**-language C++11**) now maps IDL int64 and uint64 to C++'s int64_t and uint64_t. Previously they mapped to a non-standard type, rti::core::int64 and rti::core::uint64, which on some platforms didn't exactly correspond to the int64_t and uint64_t types.

[RTI Issue IDs CODEGENII-1438]

Chapter 6 Known Issues

6.1 Classes and Types Defined in Some .NET Namespaces Cannot be used to Define User Data Types

The name of the classes and types defined in the following .NET namespaces cannot be used to define user data types:

- System
- System::Collections
- DDS

For example, if you try to define the following enumeration in IDL:

```
enum StatusKind{
    TSK_Unknown,
    TSK_Auto
};
```

The compilation of the generated CPP/CLI code will fail with the following error message:

```
error C2872: 'StatusKind' : ambiguous symbol
```

The reason for this error message is that the enumeration StatusKind is also defined in the DDS namespace and the generated code includes this namespace using the "using" directive:

```
using namespace DDS;
```

The rational behind using the "using" directive was to make the generated code shorter and more readable.

[RTI Issue ID CODEGEN-547]

6.2 Code Generation for Inline Nested Structures, Unions, and Valuetypes not Supported

Code generation for inline nested structures, unions, and valuetypes is not supported. For example, *Code Generator* will produce erroneous code for these structures:

IDL:

```
struct Outer {
    int16 outer_short;
    struct Inner {
        char inner_char;
        int16 inner_short;
    } outer_nested_inner;
};
```

XML:

```
<struct name="Outer">
    <member name="outer_short" type="int16"/>
    <struct name="Inner">
        <member name="inner_char" type="char"/>
        <member name="inner_short" type="int16"/>
    </struct>
</struct>
```

[RTI Issue ID CODEGEN-54]

6.3 .NET Code Generation for Multi-Dimensional Arrays of Sequences not Supported

The .NET code generated by *Code Generator* for multi-dimensional arrays of sequences is not correct and will not compile.

For example:

```
struct MyStruct {
    sequence<short, 4> m1[3][2];
};
```

[RTI Issue IDs CODEGENII-317, CODEGEN-376]

6.4 Request and Reply Topics Must be Created with Types Generated by Code Generator—C API Only

When using the C API to create Request and Reply Topics, these topics must use data types that have been generated by *Code Generator.* Other APIs support using built-in types and DynamicData types.

[RTI Issue ID BIGPINE-537]

6.5 To Declare Arrays as Optional in C/C++, They Must be Aliased

When generating C or C++ code, arrays cannot be declared as optional unless they are aliased.

[RTI Issue ID CODEGEN-604]

6.6 Error Generating Code for Type whose Scope Name Contains Module Called "idl"

When generating code for a file that has a member whose scope contains a module called "idl," *Code Generator* will report an error and will not generate code.

For example, *Code Generator* will not generate code for IDL with a module called "idl" such as this:

```
module idl {
    struct test{
        int32 m3;
    };
};
struct myStruct {
    idl::test m4;
};
```

The above produces this error:

```
Foo.idl line 11:4 no viable alternative at character ':'
ERROR com.rti.ndds.nddsgen.Main Foo.idl line 11:1 member
type 'dl::test' not found
```

The workaround for this issue is to prepend an underscore character ('_') to the idl module name.

[RTI Issue ID CODEGENII-661]

6.7 Examples and Generated Code for Visual Studio 2017 and later may not Compile (Error MSB8036)

The examples provided with *Connext DDS* and the code generated for Visual Studio 2017 and later will not compile out of the box if the Windows SDK version installed is not a specific number like 10.0.15063.0. If that happens, you will see the compilation error MSB8036. To compile these projects, select an installed version of Windows SDK from the Project menu -> Retarget solution.

Another option is to set the enviroment variable RTI_VS_WINDOWS_TARGET_PLATFORM_VERSION to the SDK version number. For example, set RTI_VS_WINDOWS_TARGET_PLATFORM_VERSION to 10.0.16299.0. (Note: the environment variable will not work if you have already retargeted the project via the Project menu.)

For further details, see the Windows chapter of the *RTI Connext DDS Core Libraries Platform Notes*.

[RTI Issue ID CODEGENII-800]

6.8 Invalid XSD File from an IDL/XML File if Input File Contains a Range Annotation inside a Structure and a typedef of that Structure

Code Generator generates an invalid XSD file from an IDL/XML file if the input file contains a range annotation (@min, @max, @range) inside a structure (struct/valuetype/union) and a typedef of that structure

For example, consider the following IDL file:

```
module M1 {
        struct VT1 {
                @min(0)
                int32 vt1_m1;
        };
};

typedef M1::VT1 myVT1;
```

This IDL file generates the following XSD file, which cannot be validated because the myVT1 complexType contains the same elements as its base M1.VT1, and that's not compliant with the XSD grammar:

```
<xsd:schema ...>
  <xsd:complexType name= "M1.VT1">
    <xsd:sequence>
                <xsd:element name="vt1_m1" minOccurs="1" maxOccurs="1">
          <xsd:simpleType>
            <xsd:restriction base="xsd:int">
              <xsd:minInclusive value="0"/>
            </xsd:restriction>
          </xsd:simpleType>
          </xsd:element>
</xsd:sequence>
</xsd:complexType>
 <!-- @struct true -->
  <xsd:complexType name="myVT1">
    <xsd:complexContent>
      <xsd:restriction base="tns:M1.VT1">
        <xsd:sequence>
                      <xsd:element name="vt1_m1" minOccurs="1" maxOccurs="1">
              <xsd:simpleType>
                <xsd:restriction base="xsd:int">
                  <xsd:minInclusive value="0"/>
                </xsd:restriction>
              </xsd:simpleType>
              </xsd:element>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
```

```
</xsd:schema>
```

If you try to use the generated XSD file, *Code Generator* will fail to validate the XSD file and throw one of the following errors:

```
ERROR com.rti.ndds.nddsgen.xml.XSDParser File couldn't be validated
ERROR com.rti.ndds.nddsgen.xml.XSDParser file:<...> Line: 24 Column: 33;rcase-Recurse.2: There
is not a complete functional mapping between the particles.
```

```
ERROR com.rti.ndds.nddsgen.xml.XSDParser File couldn't be validated
ERROR com.rti.ndds.nddsgen.xml.XSDParser file:///<...> Line: 16 Column: 33;rcase-
NameAndTypeOK.7: The type of element 'vt1_m1', 'null', is not derived from the type of the base
element, 'null'.particles.
```

The workaround for this issue is to disable XSD validation in *Code Generator* by enabling the option -disableXSDValidation.

Note: If the structure doesn't contain any range annotations, the generated XSD file will be validated.

[RTI Issue ID CODEGENII-1217]

6.9 Warnings when Compiling Generated Code for Traditional C++ with -O3 flag and IDL Contains FlatData types

Some C++ compilers will generate -Wmaybe-uninitialized warnings when compiling traditional C++ code (-language C++) with the compiler's -O3 optimization, if the IDL file contains a FlatData type with multiple sequences using FlatData, such as:

```
@language_binding(FLAT_DATA)
@mutable struct test {
    sequence<char, 3> myCharSeq;
    sequence<uint16, 7> myUnsignedShortSeq;
};
```

To avoid this warning, you can define **RTI_FLAT_DATA_CXX11_RVALUE_REFERENCES**. This preprocessor option turns on C++11 rvalue references in the FlatData headers, disabling a pre-C++11 workaround where the warning occurs. You can define this option through the gcc command line (**-DRTI_FLAT_DATA_CXX11_RVALUE_REFERENCES**) or at the beginning of the type implementation file (if the IDL is Foo.idl, this file is Foo.cxx).

This warning doesn't not occur when generating code for the Modern C++ API (**-language C++11**).

[RTI Issue ID CODEGENII-1327]

6.10 Recursive Structures not Supported

The OMG 'Interface Definition Language' specification, version 4.2 allows forward declarations to implement recursion: "Structures may be forward declared, in particular to allow the definition of recursive structures." While *Connext DDS* supports forward declarations, it does not currently support recursive structures.

[RTI Issue ID CODEGENII-1411]

6.11 Code Generator Server Cannot be Parallelized

Each execution of *Code Generator* server is attached to a port where it receives requests, and it can only generate code for one request at a time. Therefore, if you try to send multiple requests simultaneously, *Code Generator* server will process them sequentially.

[RTI Issue ID CODEGENII-1453]

6.12 Standalone Types not Supported

Standalone types are not supported in *Code Generator* for Modern C++ (**-language C++11**).

[RTI Issue ID CODEGENII-1412]

6.13 uint8 and int8 Types not Fully Supported

Code Generator allows IDL and XML definitions that include uint8 and int8 types. Note, however, that the *Connext DDS* language bindings use the same underlying, native 8-bit integer to represent both types. This native 8-bit integer is signed or unsigned depending on the language binding. For example:

- In Java and .NET, both uint8 and int8 map to a byte, which is signed.
- In C++, both uint8 and int8 map to a uint8, which is unsigned.
- In C and traditional C++, both uint8 and int8 map to an "unsigned char".

[RTI Issue ID CORE-10797]

6.14 64-bit Discriminator Values Greater than (2^31-1) or Smaller than (-2^31) Supported only in Java, no Other Languages

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++
- New .NET
- 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.

For example, this is not supported:

```
union union_uint64 switch (uint64) {
    case 0x100000000:
              char m_char;
    case 0x200000000:
              int32 m_int32;
    case 0x300000000:
              string<5> m_string;
};
```

This is supported:

```
union union_uint64 switch (uint64) {
    case 1:
              char m_char;
    case 2:
              int32 m_int32;
    case 3:
              string<5> m_string;
};
```

[RTI Issue ID CORE-11437]

Chapter 7 Limitations

7.1 XSD Limitation: Struct with Inheritance can't have Member with Same Name as a Member in Parent

In an IDL file, it is possible for a struct with inheritance to have a member with the same name as a member of its parent, for example:

```
struct MutableV1Struct {
    string m2; //@key
}; //@Extensibility MUTABLE_EXTENSIBILITY

struct MutableV3Struct : MutableV1Struct {
    int32 m2;
}; //@Extensibility MUTABLE_EXTENSIBILITY
```

The translation of that to XSD would generate invalid XSD because it does not allow having two members with the same name. You would see the following error mesage:

> "Elements with the same name and same scope must have same type"

Example invalid XSD:

```
<xsd:complexType name="XTypes.MutableV1Struct">
    <xsd:sequence>
        <xsd:element name="m2" minOccurs="1" maxOccurs="1"
         type="xsd:string"/>
        <!-- @key true -->
    </xsd:sequence>
</xsd:complexType>

<!-- @extensibility MUTABLE_EXTENSIBILITY -->
<xsd:complexType name="XTypes.MutableV3Struct">
    <xsd:complexContent>
        <xsd:extension base="tns:XTypes.MutableV1Struct">
            <xsd:sequence>
                <xsd:element name="m2" minOccurs="1"
                 maxOccurs="1" type="xsd:int"/>
```

```
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

If you need to generate code from invalid XSD such as seen above, you can run *rtiddsgen* with the **-disableXSDValidation** option to skip the validation step.

[RTI Issue ID CODEGENII-490]

7.2 Generated Code for Nested Modules in Ada May Not Compile

Code Generator follows the Object Management Group (OMG) IDL-to-Ada specification in order to map modules:

> Top level modules (i.e., those not enclosed by other modules) shall be mapped to child packages of the subsystem package, if a subsystem is specified, or root library packages otherwise. Modules nested within other modules or within subsystems shall be mapped to child packages of the corresponding package for the enclosing module or subsystem. The name of the generated package shall be mapped from the module name.

The generated code produced by following this specification does not compile when referencing elements from a nested module within the top-level module, as shown in the following example:

```
module Outer
{
        module Inner
        {
                struct Structure
                {
                        int32 id;
                };
        };

        struct Objects
        {
                Inner::Structure nest;
        };
};
```

This failure to compile happens because Ada does not allow a parent package to reference definitions in child packages.

[RTI Issue ID CODEGENII-813]

7.3 Mixing Different Versions of Code Generator Server is not Supported

If you run different versions of **rtiddsgen_server** in the same port, the generated code could be generated by a different version than the one expected. **rtiddsgen_server** starts a server that generates code. If this server is still up when you run another version of **rtiddsgen_server**, your code is generated by the server that was already up, which is a different version than the one you wanted.

For example, if you run *Code Generator* server 2.5.0, then in the same port you run *Code Generator* server 3.0.1, *Code Generator* 2.5.0 might generate your code when you wanted *Code Generator* 3.0.1 to generate it.