# RTI Code Generator

## Release Notes

## Version 4.2.0

## Trademarks



## Copy and Use Restrictions



## Third-Party Software

RTI software may contain independent, third-party software or code that are subject to third-party license terms and conditions, including open source license terms and conditions. Copies of applicable third-party licenses and notices are located at community.rti.com/documentation. IT IS YOUR RESPONSIBILITY TO ENSURE THAT YOUR USE OF THIRD-PARTY SOFTWARE COMPLIES WITH THE CORRESPONDING THIRD-PARTY LICENSE TERMS AND CONDITIONS.

## Notices

*Deprecations and Removals*

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

*Deprecated* means that the item is still supported in the release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in a release, RTI hereby provides customer notice that RTI reserves the right after one year from the date of such release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

**Technical Support**
Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: https://support.rti.com/

# Contents

# 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 table in the Supported Platforms section of the RTI Connext Core Libraries Release Notes by using the script *rtiddsgen*.
- As a native application, *Code Generator* is supported on all host platforms listed in the table in the Supported Platforms section of the RTI Connext Core Libraries Release Notes by using the script *rtiddsgen_server*.

# Chapter 2 Compatibility

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

*Code Generator* has been tested with AdoptOpenJDK 17.0.6, which is included in the installation package.

# Chapter 3 What's New in 4.2.0

## 3.1 Code Generator Will Not Parse Duplicated XML Files

Previously, *Code Generator* would display an error if you generated code for the following scenario:

You have three XML files: `A.xml`, `B.xml` and `C.xml`.

`A.xml` includes `B.xml` and `C.xml`.

`B.xml` includes `C.xml`.

If you generated code for `A.xml`, you would get an error that the types from `C.xml` are duplicated. If the files were IDL, this could be resolved by using `#ifdef` or `#ifndef` clauses, but these are not available in XML.

*Code Generator* now checks if an XML file has already been included and will not parse it again. You won't receive a type duplication error in the above scenario.

## 3.2 Added Warning to Code Generator when Defining Multiple Enums with Common Enumerator in the Same Namespace

The OMG 'Interface Definition Language' specification, version 4.2 does not allow the use of a common enumerator in multiple enums in the same namespace. For example, the following IDL is not compliant with the IDL 4.2 specification:

```
enum ENUM1 {A,B,C};
enum ENUM2 {C,D,E}; // C is a common enumerator in both enums.
```

Previously, *Code Generator* would not display a warning if it generated code for the above IDL, even though the IDL was not compliant with the OMG specification.

Now, if you try to generate code for the above IDL, *Code Generator* will display a warning, but will still generate the code. If the flag `-strict` is used (to enable strict OMG specification compliance checks) and you try to generate code for the above IDL, *Code Generator* will fail and report that there is a common enumerator in multiple enums.

> **NOTE:** If you are generating code for C or C++98 for an IDL with a common enumerator in multiple enums, you must use the flag `-qualifiedEnumerator`, or else code generation will fail.

## 3.3 Added Java Exit Code to Code Generator

If a Java program does not end with an explicit `System.exit(0)` command, it might continue running inside a Docker™ container or virtual machine. This improvement ends *Code Generator* with an

explicit `System.exit(0)`.

## 3.4 Added Flag to Display Type Sizes

The `-typeSizes` flag has been added to *Code Generator*. This flag will display the following types information:

- Maximum key serialization size.
- Minimum serialization size.
- Maximum serialization size.

If the type sizes can be computed, *Code Generator* will display the sizes. In some scenarios, however, *Code Generator* will not be able to compute the type sizes and will display the following:

- If there is a recursive type, *Code Generator* will display `Undefined (Recursive Type)`.
- If there is a type that is unresolved because you are using `@resolve-name false`, *Code Generator* will display `Undefined (Unresolved Member)`.
- If the type is bigger than the maximum serialized size, *Code Generator* will display `Error (Over Max Serialized Size)`.

As an example, for the following IDL:

```
union typeUnion switch(boolean){
    case TRUE: int16 m1;
    case FALSE: int64 m2;
};

struct typeStruct {
    @optional int32 m1;
    typeUnion m2;
};
```

If you call *rtiddsgen* with the `-typeSizes` flag:

```
rtiddsgen -typeSizes idlFile.idl
```

*Code Generator* will produce the following output:

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 4.1.0, please wait ...
INFO com.rti.ndds.nddsgen.dataRepresentation.CDR
Xcdr1 types sizes from idlFile.idl:
+=============================================================================+
| Type Name  | Max Key Serialized Size | Min Serialized Size | Max Serialized Size |
+=============================================================================+
| typeUnion  | 16                      | 4                   | 16                  |
+-----------------------------------------------------------------------------+
| typeStruct | 28                      | 8                   | 28                  |
```

```
+-------------------------------------------------------------------------------+

INFO com.rti.ndds.nddsgen.dataRepresentation.CDR
Xcdr2 types sizes from idlFile.idl:
+===============================================================================+
| Type Name  | Max Key Serialized Size | Min Serialized Size | Max Serialized Size |
+===============================================================================+
| typeUnion  | 16                      | 8                   | 16                  |
+-------------------------------------------------------------------------------+
| typeStruct | 28                      | 16                  | 28                  |
+-------------------------------------------------------------------------------+

INFO com.rti.ndds.nddsgen.Main Done
```

**NOTE:** Using the flag may reduce *Code Generator* performance. We recommend you disable the flag if you just want to generate code and you don't want or need type information.

## 3.5 Added Support to Generate Examples in C# for .Net 8

*Code Generator* can now generate example C# files for .Net 8 with the command-line argument `-platform net8`.

## 3.6 New Command-Line Arguments to Define the Endianness and Data Representation

*Code Generator* has two new command-line arguments to specify the endianness and the data representation. These will optimize *DomainParticipant* creation for a specific endianness and data representation. The arguments are only valid in C, C++98, and C++11.

**-allowedDataRepresentation:** Generates code for just one data representation: XCDR1 or XCDR2.

**-allowedEndian:** Defines the endianness for the generated code: bigEndian or littleEndian.

## 3.7 Added Support for New OMG IDL4 to C++ Language Mapping

*Code Generator* provides a new mapping for Modern C++, based on the new OMG IDL4 to C++ Language Mapping standard (IDL4-CPP).

To generate code for the new mapping, use the following command-line options:

```
-language C++11 -standard IDL4_CPP
```

The most notable changes in the IDL4-CPP mapping are:

- IDL structs map to C++ structs with public fields, instead of classes with getters and setters
- IDL unions still map to classes with getters and setters with the following additions:
  - For members selected by multiple labels, a setter receiving the discriminator value as a second argument is also generated
  - A method called `_default()` that sets the union to its default discriminator is generated
- string and wstring constants map to `std::string_view` and `std::wstring_view` for platforms that support C++17

## 3.8 Added Support for rtiddsgen_server on macOS

*RTI Code Generator's* **rtiddsgen_server** is now supported on macOS. Previously, it was supported only on Windows and Linux. This support was added in release 7.1.0, but not documented at that time.

## 3.9 Third-Party Software Upgrades

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

| Third-Party Software | Previous Version | Current Version |
|---|---|---|
| AdoptOpenJDK | 11.0.13 | 17.0.6 |
| AdoptOpenJRE | 11.0.13 | 17.0.6 |

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

# Chapter 4 What's Fixed in 4.2.0

## 4.1 Fixes Related to Generated Code (Multiple Languages)

### 4.1.1 Include flag used without a space caused Code Generator to fail

When using the flag `-I` without a space between the flag and the included folder, *Code Generator* failed, reporting that it was not supported. This issue was introduced in *Code Generator* 4.1.0 (*Connext* 7.1.0). This issue is fixed; you can successfully use the `-I` flag without a space (for example, `-I./myInclude/`).

[RTI Issue ID CODEGENII-1867]

### 4.1.2 Identifiers collision detection was not case insensitive when using strict

The IDL spec says:

```
When comparing two identifiers to see if they collide:
--Upper- and lower-case letters are treated as the same letter. Table 7-2 defines the
  equivalence mapping of upper- and lower-case letters.
--All characters are significant.

Identifiers that differ only in case collide, and will yield a compilation error under
certain circumstances. An identifier for a given definition must be spelled identically
(e.g., with respect to case) throughout a specification.
```

And for the keywords it says:

```
Identifiers that collide with keywords are illegal.
```

This behavior was not implemented in *Code Generator*. *Code Generator* did not fail if you generated code for the following IDL, in which there is a collision between two identifiers and another collision between an identifier and a keyword.

```
struct mystruct {
long BOOLEAN; // Collision with the keyword "boolean"
};
struct MYSTRUCT { // Collision with the identifier "mystruct"
long m1;
}
```

You can now get *Code Generator* to fail when these types of collisions occur in an IDL file, by using the command-line option `-strict`.

[RTI Issue ID CODEGENII-1237]

### 4.1.3 int8 constants were not supported because type was mapped and parsed as an unsigned value

Support for int8 was incomplete in *Code Generator*. As a result, unexpected warnings and errors may have occurred.

The issue is fixed; int8 constants compile and work as expected.

[RTI Issue ID CODEGENII-1495]

### 4.1.4 Code Generator did not fail if discriminator of a union was a 64-bit integer

As documented in the known issue 64-bit Discriminator Values Greater than (2^31-1) or Smaller than (-2^31) Supported only in Java, no Other Languages, using a 64-bit integer with or without a sign as a union discriminator is not supported. However, *Code Generator* generated code without errors.

To avoid possible issues in production code, *Code Generator* will now detect and fail if the discriminator of a union is a 64-bit integer.

[RTI Issue ID CODEGENII-1864]

### 4.1.5 Code not stored in specified output directory on Windows when IDL was in a Symlink

In Windows environments, when generating code using a specified output folder for an IDL located in a Symlink folder, the code was stored in the Symlink folder instead of the specified output directory. This issue was caused by a JDK bug. We have applied a workaround; the generated code is now stored in the ouput directory when specified.

[RTI Issue ID CODEGENII-1891]

## 4.2 Fixes Related to Generated Code (C#)

### 4.2.1 Generated code did not compile when a negative number was assigned to an int8 constant in C#

When a negative value was assigned to an int8 constant in C#, compilation errors occurred.

This issue is fixed; int8 constants are now fully supported in C#.

[RTI Issue ID CODEGENII-1593]

### 4.2.2 C# generated examples may not have used latest patch available

The example code generation for C# defined a vanilla release version number for C# libraries instead of the version of the latest patch installed. This issue is fixed. The generated example code will now use the latest patch installed for the current version of the product.

[RTI Issue ID CODEGENII-1850]

### 4.2.3 C# Copy Constructor did not create a deep copy of alias of collections

The C# Copy Constructor created a shallow copy of the alias of collections. This issue is fixed; C# Copy Constructor now does a deep copy of all members in an aggregated type.

[RTI Issue ID CODEGENII-1895]

## 4.3 Fixes Related to Generated Code (Java)

### 4.3.1 Java serialized sample min or max size may have returned an incorrect value for union mutable types using XCDRv1 encoding

When *Code Generator* generated Java code for a mutable union, the methods `get_serialized_sample_max_size` and `get_serialized_sample_min_size` from the class `TTypeSupport` may have returned an incorrect value, when using XCDRv1. This issue occurred because *Code Generator* was not adding the sentinel to the actual current alignment with the biggest or smallest member size of the union, resulting in an incorrect alignment. This issue is now fixed.

[RTI Issue ID CODEGENII-1820]

### 4.3.2 Java serialized sample size may have returned an incorrect value for union mutable types using XCDRv1 encoding

When *Code Generator* created Java code for a mutable union based on an enum, the method `get_serialized_sample_size` may have returned an incorrect value. This error occurred because *Code Generator* incorrectly determined that the header for the union's member needed an extended Id, though it may not have been needed. This issue is fixed.

[RTI Issue ID CODEGENII-1825]

### 4.3.3 Min and max size may have returned incorrect values for union mutable types using XCDR2 encoding

The methods `get_serialized_sample_max_size` and `get_serialized_sample_min_size` may have returned an incorrect value when *Code Generator* generated Java code for union mut-

able types using XCDR2. This issue was caused by a problem with the `LC` value that was being used for the header of the union's members. This issue is now fixed.

[RTI Issue ID CODEGENII-1828]

### 4.3.4 Incorrect header size for get_serialized_key_max_size in java generated code

When generating Java code for a mutable type, *Code Generator* has to add a header for each of its members. The size of the header varies depending on whether XCDR1 or XCDR2 is used; however, *Code Generator* always added the same header size to `get_serialized_key_max_size`. This issue occurred only when using XCDR2 and mutable types. This is now fixed.

[RTI Issue ID CODEGENII-1873]

### 4.3.5 Incorrect max serialized size value in generated Java code for wstrings when using XCDR2

*Code Generator* incorrectly added a terminating Null character in the method `get_serialized_sample_max_size` when generating Java code for XCDR2. As a result, the max serialize size value returned by this method was incorrect. For example, the following struct should return the max serialize size 16, but *Code Generato*r calculated 18.

```
struct PluginStruct {
wstring<4> member1;
};
```

This issue is fixed; the `get_serialized_sample_max_size` method returns the correct size for XCDR2.

[RTI Issue ID CODEGENII-1887]

### 4.3.6 Java type suffix not added to hexadecimal constant values

When *Code Generato*r created Java code for a long long constant value, and the value was specified as a hexadecimal, the generated code did not add the suffix `L` to specify that the value is a Long value. This is now fixed.

For example, for this idl:

```
const long long LONG_CONST   = 0x7fffffffffffff;
```

*Code Generator* now generates:

```
public class LONG_CONST {
public static final long VALUE = 0x7fffffffffffffL;
}
```

[RTI Issue ID CODEGENII-1905]

# 4.4 Fixes Related to Generated Code (Python)

## 4.4.1 Generated Python code produced syntax error for types in a different directory

When an IDL had an import to another IDL that was in a different directory, and *Code Generator* generated code for that IDL for Python, the generated code produced a syntax error. This is now fixed; *Code Generator* produces correct code when an IDL is imported from another directory.

[RTI Issue ID CODEGENII-1847]

## 4.4.2 Topics for types inside a module in a Connext Python application may not have communicated with other Connext applications

A *Connext* Python application may not have communicated by default with a *Connext* application in another programming language if the following conditions were true:

- The types were defined inside a module
- The type information was not propagated on the wire, which caused the type matching algorithm to default to type-name-based matching instead of type-compatibility-based matching

This problem was caused by an inconsistent registered type name for Python topics. For example, a type `Foo` inside a module `Bar` was registered by default as `Bar_Foo` instead of `Bar::Foo`. This problem has been resolved.

If you experienced this problem, you need to re-generate your Python types from your IDL types.

[RTI Issue ID CODEGENII-1866]

## 4.4.3 Python keywords used in IDL were not prefixed

*Code Generator* did not prefix IDL names that conflicted with Python keywords. For example, for the following IDL, errors occurred because *Code Generator* did not prefix the if name in the Python code:

```
struct if {
long l;
};
```

The issue is fixed; IDL names are now prefixed if they clash with Python keywords.

[RTI Issue ID CODEGENII-1906]

# 4.5 Other Fixes

## 4.5.1 Code Generator did not fail for optional sequences in Ada

*Code Generator* did not fail when generating code for an optional sequence in Ada, even though optional sequences are not supported in Ada. Now, *Code Generator* will fail if the IDL/XML/XSD file contains an optional sequence.

For example, this IDL will now fail for Ada:

```
struct MyStruct {
@optional
sequence<short> m1;
};
```

This is a known issue that will be fixed in a future release. For now, the workaround is to use an empty sequence to emulate an unset optional.

[RTI Issue ID CODEGENII-1666]

## 4.5.2 Command-line option tips not printed if you entered an invalid option

In release 7.1.0, if you entered an invalid *Code Generator* command-line option, *Code Generator* did not print out the helpful list of valid command-line options. This regression has been fixed.

[RTI Issue ID CODEGENII-1868]

# Chapter 5 Previous Releases

## 5.1 What's New in 4.1.0

### 5.1.1 New Command-Line Option for Generating Included Files

This release adds the command-line option **-generateIncludeFiles**. With this option, *Code Generator* generates code for any included file in the inputs.

For example:

```
rtiddsgen -language python Foo.idl -generateIncludeFiles
```

Imagine you have the following two files:

```
// File Bar.idl

struct Bar {
...
};
```

```
// File Foo.idl
#include "Bar.idl"

struct Foo {
    Bar b;
};
```

This example will produce the following files:

- Foo.py
- Bar.py

### 5.1.2 Ability to Generate Code for Multiple IDL/XML/XSD Files at the Same Time

*Code Generator* can now receive multiple input files for code generation. For example:

```
rtiddsgen -language C -create typefiles hello_world1.idl hello_world2.idl
```

This new feature also allows passing one or more directories as input. To use folders as inputs, use one of the following command-line options: **-inputIDL**, **-inputXML**, or **-inputXSD**. *Code Generator* will scan the folder and generate code for the files with the extension indicated by the input flag.

```
rtiddsgen -language C -create typefiles -inputIDL folder folder2
```

The new command-line option **-r** will activate the recursive scan of all the input folders. See Specifying Multiple Input Files in the Code Generator User's Manual.

### 5.1.3 Added Support for Generated Types Without Connext in Modern C++ (Standalone Mode)

The type code generated for C++11 can now be used without linking with *Connext* libraries. Standalone code for C++11 works in a similar way as C or C++98.

### 5.1.4 Added Support for @topic and @default_nested Annotations

The annotations @topic and @default_nested have been added to *Code Generator*. See Using Builtin Annotations in the Core Libraries User's Manual for more information.

### 5.1.5 Added Support for Unbounded Sequences and Strings in Ada

*Code Generator* now allows the use of the flag **-unboundedSupport** with the Ada language. This flag activates unbounded support for strings and sequences.

### 5.1.6 Added Support to Generate Examples in C# for .Net 7

*Code Generator* can now generate example C# files for .Net 7. To do so, run **rtiddsgen** with the command-line argument **-platform net7**.

### 5.1.7 Create Advanced Examples in Python

Previously, you could use the command-line argument **-exampleTemplate advanced** to create advanced examples for all languages except C and Python. Now you can use the **-exampleTemplate advanced** argument for Python, too. The advanced example uses an asynchronous generator to read over samples as they are received and monitors status updates in the *DataWriter* and *DataReader*. See the Advanced Example section in the RTI Code Generator User's Manual for more information.

### 5.1.8 Added Support to Code Generator for Loading Templates Containing Macros

You can now define and load macros for use by *Code Generator* templates. To add these templates to a specific language (`<lang>`), create the following folder:

```
<NDDSHOME>/resource/app/app_support/rtiddsgen/templates/<lang>/macros/
```

All the templates that end in **.vm** that you add to the macros folder will be loaded. You can then call these macros from any template in `<NDDSHOME>/resource/app/app_support/rtiddsgen/templates/<lang>/`, such as **type.vm**. See Customizing the Generated Code in the Code Generator User's Manual for more information.

**NOTE:** This feature is supported by both *Connext Professional* and *Connext Micro*.

### 5.1.9 New Way to Initialize Arrays in C++11 Generated Code

Previously, *Code Generator* initialized the arrays in the constructor using **rti::core::fill_array**. Now, *Code Generator* initializes them using aggregate initialization in the member's declaration instead of using **rti::core::fill_array** in the constructor.

## 5.2 What's Fixed in 4.1.0

### 5.2.1 Fixes Related to C, Traditional C++, and Modern C++ Generated Code

#### 5.2.1.1 Aliases and Arrays Not Correctly Initialized in Unions

In previous *Code Generator* releases, aliases and arrays were not correctly initialized in the union's constructors. This issue is now resolved; *Code Generator* now initializes aliases and arrays using aggregate initialization in the member's declaration, instead of using `rti::core::fill_array` in the constructor.

[RTI Issue ID CODEGENII-842]

#### 5.2.1.2 Compile-Time Error when Using -constructor with Types Containing Sequence of Pointers

After 6.0.1, if you created an IDL with a sequence of pointers, and generated code for C++98 with the **-constructor** flag, you encountered a compile-time error, because *Code Generator* was assigning NULL to the sequence. This problem has been fixed. The code now generates the constructor correctly when using a sequence of pointers.

[RTI Issue ID CODEGENII-1596]

#### 5.2.1.3 Typo in a Condition in ndds_standalone_type.h

In the file **<NDDSHOME>/resource/app/app_support/rtiddsgen/standalone/include/ndds_standalone_type.h**, a preprocessor condition incorrectly checked if the variables **RTI_LINUX** or **RTI_DARWIN** were defined. This condition is now fixed.

[RTI Issue ID CODEGENII-1657]

#### 5.2.1.4 Generated Code for C++11 Didn't Compile When Using @external int8/uint8/octet

When generating code for C++11, if a type had a member that was an external int8, uint8, or octet, the code was generated correctly, but the compilation failed due to an invalid cast. This problem has been fixed. Now the generated code will compile and work as expected.

[RTI Issue ID CODEGENII-1727]

#### 5.2.1.5 C++11 Examples Fail When Using External Annotation

Examples generated for C++11 for a type with external members would fail if used directly. This was because a sample with an external could not be sent with a null value. This issue has been fixed by initializing the first-level external members.

Note that higher-level external members will still cause examples to fail, such as a type with a member of another type that has an external. In this case, the external must be initialized by the user.

[RTI Issue ID CODEGENII-1747]

#### 5.2.1.6 Forward Declaration Did Not Work for C and C++98 When Using Incomplete Type in a Sequence

If you generated code for C or C++98 for an IDL that contained a forward declaration, and you used the forward-declared type in a sequence before defining it, the code failed at compilation time:

```
struct MyStructFD;

struct MyStruct{
sequence <MyStructFD> member_1;
};
struct MyStructFD {
long member_FD;
};
```

This problem is now fixed. If you generate code for the example above, it will generate correctly and compile. For more information about forward declarations, see IDL Forward Declaration in the RTI Connext Core Libraries Users Manual.

[RTI Issue ID CODEGENII-1807]

#### 5.2.1.7 DDS Topic Type Name Value May Return a Different Value Than the One Used in the typeCode

If you used a keyword as the name of a type or a module, **dds::topic::topic_type_name<T>::value()** would return a different value than the one used in the typeCode. This issue has been fixed. Now, **dds::topic::topic_type_name<T>::value()** and the type code are populated with same type name.

[RTI Issue ID CODEGENII-1817]

### 5.2.2 Fixes Related to C# Generated Code

#### 5.2.2.1 Copy Directive (@copy) Was Not Available for C#

The following directives are now functional when generating code with *Code Generator* for C# :

```
//@copy "//copy this message for all the languages"
//@copy-declaration "//copy this message for all the languages, just where the types are
being declared"
```

```
//@copy-cs "//copy this message just for C#"
//@copy-cs-declaration "//copy this message just for C#, just where the types are being
declared"
```

[RTI Issue ID CODEGENII-1502]

#### 5.2.2.2 Compilation Error When Using copy-c Directive Inside of a Structure in C#

Previously, if *Code Generater* generated code for C# from an IDL in which there was a type with a **//@copy** directive inside, the code would be generated, but not compile. Now, the generated code compiles, and the **//@copy** directives are available in C#.

[RTI Issue ID CODEGENII-1713]

#### 5.2.2.3 C# Generated Code Does Not Compile When a Sequence or Array is Named hash

In the generated code for an IDL, a type with a member called **hash** would conflict with the local variable **hash** in the method **GetHashCode()**. This has been fixed by adding **this.** to **hash** when referring to the name of a member inside the method **GetHashCode()**.

[RTI Issue ID CODEGENII-1714]

#### 5.2.2.4 Incorrect C# Generated Code When a Union Based on the Alias of an enum is in a Different Module from the enum

If you generated code for C# for an IDL with a union that was based on an enum defined in a different module, the code would not compile. This issue has been resolved.

[RTI Issue ID CODEGENII-1797]

#### 5.2.2.5 Negative enums Produced Compilation Error in C#

The generated code for an enumerator type may have caused a compilation error in C#. This happened when one of the values was negative and the enumerator was a member of an aggregated type:

```
enum MyStatus { ProblemEnum = -1, Correct = 0};
```

The C# compiler complained that the assignation of the negative value must be between parentheses.

This issue is fixed. Now, the generated code is correct and will compile.

[RTI Issue ID CODEGENII-1799]

### 5.2.3 Fixes Related to Java Generated Code

#### 5.2.3.1 Incorrect Values for Serialized Sample Max and Min Size in Java Generated Code

When *Code Generator* generated Java code for an IDL with mutable types or types with optional members, you could get an incorrect value for the following methods:

- `get_serialized_sample_max_size`
- `get_serialized_sample_min_size`

This issue has been resolved.

[RTI Issue ID CODEGENII-655]

### 5.2.3.2 Issue With Java Generated Code When Using Unbounded Support Flag

When generating code for Java using the flag for unbounded support (**-unboundedSupport**), there was an issue with the generated USER_QOS_PROFILES.xml. The property **dds.data_writer-.history.memory_manager.fast_pool.pool_buffer_max_size** was contained in a pair of **<value>** tags, and the properties **dds.data_writer.history.memory_manager.java_stream.min_size** and **dds.data_writer.history.memory_manager.java_stream.trim_to_size** were in another pair. This is now fixed; all the properties are contained in the same group of **<value>** tags.

[RTI Issue ID CODEGENII-1631]

### 5.2.3.3 Possible Data Loss Deserializing a Union with an enum Based Discriminator When dds.sample_assignability.accept_unknown_enum_value is 1

When the property **dds.sample_assignability.accept_unknown_enum_value** was set to 1 and the type was an appendable union, data could be lost or corrupted during deserialization.

For the following example:

```
enum ColorV1 {
    RED_1
};

// Subscriber
@interpreted(true)
union ColorUnionV1 switch (ColorV1) {
    case RED_1:
        long m1;
};

@interpreted(true)
enum ColorV2 {
    RED_2,
    GREEN_2
};

// Publisher
@interpreted(true)
union ColorUnionV2 switch (ColorV2) {
    case RED_2:
        long m1;
    case GREEN_2:
```

```
        octet m3;
};
```

If a subscriber that expected a **ColorUnionV1** received a **ColorUnionV2** sample with **ColorV2.GREEN_2** as discriminator, an exception would be thrown because the subscriber would try to read **ColorUnionV1.m1**, which is bigger than the data in the wire, **ColorUnionV2.m3**.

The issue has been fixed. Now, if a subscriber that expects a **ColorUnionV1** receives a **ColorUnionV2** with the discriminator set to **ColorV2.GREEN_2**, the received **ColorUnionV1** sample will be:

- Set to the default value if **dds.sample_assignability.accept_unknown_union_discriminator** is 1, or;
- Dropped if **dds.sample_assignability.accept_unknown_union_discriminator** is 0.

[RTI Issue ID CODEGENII-1823]

## 5.2.4 Fixes Related to Python Generated Code

### 5.2.4.1 Type idl.int8 Generation for Python Inconsistent With Other APIs

The type generated for octet, uint8, and int8 for Python was **idl.int8**, but this caused inconsistency with other APIs in which this type is not supported. Now *Code Generator* generates the type **idl.uint8** for octet, uint8, and int8.

[RTI Issue ID CODEGENII-1753]

### 5.2.4.2 Incorrect Generated Python Code if a Union Based on an enum Didn't Have the Default Enumerator, in a Branch of the enum

If an IDL had a union based on an enum, and this union didn’t have a branch with the default enum value, the Python-generated code was not correct. Now, the generated code is correct, even if you have a union without the default enum value.

[RTI Issue ID CODEGENII-1771]

### 5.2.4.3 @allowed_data_representation Annotation Not Processed Correctly in Python

The annotation @allowed_data_representation ignored the parameter passed in the IDL/XML and always produced the same generated code in the Python decorator.

```
idl.allowed_data_representation(xcdr2=False, xcdr1=True)
```

This release fixes the problem. Now, the generated code in the decorator will match the parameter passed to the annotation @allowed_data_representation in the IDL/XML.

[RTI Issue ID CODEGENII-1773]

#### 5.2.4.4 Python Generated README.txt When It Should Not Have

*Code Generator* always generated the README.txt file for Python even when it should not have. It should have generated the README.txt file only when the **-example <arch>** or **-create <makefiles>** command-line argument was used. Now, README.txt won't be generated for Python unless the **-example <arch>** or **-create <makefiles>** command-line argument is used.

[RTI Issue ID CODEGENII-1775]

### 5.2.5 Fixes Related to Generated Code (Multiple Languages)

#### 5.2.5.1 Code Generator Allowed Setting Scoped Names or Negative Values as a Size

*Code Generator* allowed setting scoped names and negative values as a size, for an array, string, or sequence, resulting in a compilation time error. Now, if something other than a positive integer is used as a size, *Code Generator* fails.

[RTI Issue ID CODEGENII-407]

#### 5.2.5.2 Invalid Generated Code When a Type Was Inside a Set of Modules With Repeated Names

*Code Generator* would generate code for C++98, C++11 and C# that did not compile when the following occurred:

- There was a type inside a set of modules, and;
- Two of those modules had the same name.

For example:

```
module A{
    module B{
        module A{
            struct myStruct {
                long m1;
            };
        };
    };
};
```

This issue has been corrected for all affected languages.

[RTI Issue ID CODEGENII-609]

#### 5.2.5.3 Code Generation Fails for Arrays and Sequences of Aliases with Default, Max, or Min Values

*Code Generator* does not support generating code for arrays and sequences with default, max, or min values. However, *Code Generator* would generate code for an IDL that set a default, max, or min value in an alias (i.e. typedef) and then used that alias as the base type for a sequence or array. For example:

```
@min(0)
@max(20)
@default(10)
typedef long myLongTypedef;

struct Foo {
    myLongTypedef myLongArray[2];
    sequence<myLongTypedef> myLongSequence;
};
```

In this release, *Code Generator* will fail to generate code for the above IDL the same way it would fail for a sequence with default, max, or min values.

[RTI Issue ID CODEGENII-1314]

#### 5.2.5.4 -Wsign-conversion and -Wconversion Warnings Removed from Generated Code

*Code Generator* introduced **-Wsign-conversion** and **-Wconversion** warnings in the generated code. All of these warnings have now been removed from the generated code.

[RTI Issue ID CODEGENII-1633]

#### 5.2.5.5 Code Generator Did Not Allow Forward Declaration of Interfaces

*Code Generator* did not allow forward declaration of interfaces. Now it does. A forward declaration of interfaces can be added to the IDL, and the code will be generated.

[RTI Issue ID CODEGENII-1720]

#### 5.2.5.6 Code Generator Did Not Check if Base Interface Exists When Inheriting

*Code Generator* did not check if the base interface of inheritance existed; as a result, it generated code with issues. Now, if an interface inherits from an interface that doesn't exist, code generation will fail.

[RTI Issue ID CODEGENII-1744]

#### 5.2.5.7 Code Generator Allowed Inheriting from Forward-Declared Structure

The specification doesn't allow inheritance from a forward-declared structure. *Code Generator* allowed this behavior and generated code with issues silently. Now if a structure inherits from a forward-declared structure, code generation will fail.

[RTI Issue ID CODEGENII-1745]

#### 5.2.5.8 Code Generator Slowed Down When There Were Multiple Levels of Aliases

An IDL/XML file with multiple levels of aliases could cause *Code Generator* to slow down significantly, due to recursive value calculations. The issue has been fixed. Now, *Code Generator* calculates the values once and stores them to use them when necessary.

[RTI Issue ID CODEGENII-1810]

#### 5.2.5.9 Prevent Code Generation for typedefs with Inconsistent Default, Max, or Min values

Previously, you could generate code for the following IDL:

```
@max(5)
typedef long myLongTypedef;

@min(10)
typedef myLongTypedef myLongTypedef2;

struct Foo{
    myLongTypedef2 m1;
};
```

However, when these typedefs are resolved, they result in a member with a minimum value of 10 and a maximum value of 5, which is not valid. This issue has been resolved. In this release, if you try to generate code that shows inconsistencies between max, min, or default values when the typedefs are resolved, *Code Generator* will fail.

[RTI Issue ID CODEGENII-1815]

## 5.3 What's New in 4.0.0

### 5.3.1 New and Removed Platforms

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

### 5.3.2 New Python Language Binding (Experimental)

*Code Generator* can now generate code and examples for Python from IDL, XML, and XSD. To use this new binding, use the command-line option **-language Python**.

For generating examples, the only available platform for Python is “universal”. See the RTI Code Generator User's Manual for more information.

### 5.3.3 Use -language C++98 Instead of -language C++ to Generate Traditional C++ code

This release introduces the C++98 language option for traditional C++ code generation. From now on, use **-language C++98** to specify code generation for traditional C++. (Use **-language C++11** for modern C++, as before.) Using *Code Generator* without specifying a language, or specifying **-language C++**, may cause confusion since it does not specify the language standard. The use of **-language C++** or not using **-language** at all is not recommended and will generate a warning during code generation.

Note that C++98 and C++11 are the minimum C++ versions required for the traditional C++ and modern C++ APIs, respectively. Applications can use newer C++ standards.

### 5.3.4 Improve hashCode Function in Java Generated Code

Previously, the generated **hashCode** function was just the addition of all the members of the type, producing collisions. After this improvement, we have reduced the number of collisions, by using a prime number and multiplying it by each member before adding them all.

### 5.3.5 Code Generator now Fails for Optional Sequences in C#

Previously, the optional annotation was silently ignored for sequences in C#. Now, *Code Generator* will fail if the IDL/XML/XSD file contains an optional sequence.

For example, this IDL will now fail for C#:

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

This problem is a known issue that will be fixed in a future release. For now, the workaround is to use an empty sequence to emulate an unset optional. See 6.13 C# Code Generation for Optional Sequences not Supported on page 32 for details.

### 5.3.6 Deprecations and Removals

This section describes features that are *deprecated* or *removed* starting in release 7.0.0.

*Deprecated* means that the item is still supported in this release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported. By specifying that an item is deprecated in this release, RTI is hereby providing customer notice that RTI reserves the right after one year from the date of this release and, with or without further notice, to immediately terminate maintenance (including without limitation, providing updates and upgrades) for the item, and no longer support the item, in a future release.

This section serves as notice under the Real-Time Innovations, Inc. Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software.

#### 5.3.6.1 Language C++03 option removed in this release

The *rtiddsgen* option **-language C++03** was deprecated starting in release 6.1.0. Starting in 6.1.0, *Code Generator* produced a warning message during code generation that C++03 support would be removed in a future release. That removal has happened as of release 7.0.0.

For the Modern C++ API, you now must use **-language C++11**; for the Traditional C++ API, you should use **-language C++98**, although you can continue to using **-language C++**.

The Modern C++ API now requires a C++11 compiler (or newer). The Traditional C++ API continues to support C++98 compilers (or newer).

#### 5.3.6.2 Legacy C# language binding removed in this release

This release removes support for code generation for the legacy C# API and C++/CLI. Likewise, the **-dotnet** parameter (used to specify the legacy C# code generation) has also been removed. From release 7.0.0 forward, the **-language C#** command-line option will produce C# code using the latest C# API that was introduced in 6.1.0 (*rtiddsgen* 3.1.0).

## 5.4 What's Fixed in 4.0.0

### 5.4.1 Possible Memory Leak in Builtin Types after Allocation Error

If an allocation error occurred during creation of a builtin type, some of the allocated memory for internal members mapped as pointers may not have been released. This issue has been fixed. Now all the allocated memory for builtin types is released when errors occur during memory allocation.

[RTI Issue ID CODEGENII-1624]

### 5.4.2 Using Batching for Types with Optional Members may have Caused Serialization/Deserialization Errors in Java

The serialization and deserialization of samples may have caused data corruption in types with optional members when the batching feature was enabled. The errors in the communication may have caused data corruption when samples were written and may have triggered exceptions on the Subscriber side. This issue affected Java code only. Since the issue affected both the serialization and deserialization methods, interoperability with other languages may have been affected too. This problem has been resolved.

[RTI Issue ID CODEGENII-1638]

### 5.4.3 @copy Directives Resulted in Multiple Copies of Same Directive in Generated Code/Header in C++11

Copy directives (for example, copy directives related to modules) were generated multiples time, even if that didn't make sense. This problem has been resolved. Now, a copy directive will be attached to an entity. The directive will only be generated when the related entity is generated.

For example, for the following IDL:

```
//@copy-c //Generated when the module moduleWithDirective is generated
module moduleWithDirective {
    //@copy-c #ifdef something generated with Foo
    struct Foo {
        //@copy-c // generated with  longWithDirective
        long longWithDirective;
    };
    //@copy-c #endif //generated with Bar
    //@copy-c #ifdef somethingNotDefined //generated with Bar
    struct Bar {
        long myLong;
    };
    //@copy-c #endif //generated with Bar as postfix directive because the closing module
};
```

The first line, `//@copy-c //Generated when the module moduleWithDirective is generated`, belongs to moduleWithDirective and will be copied only once (modules only generate code once).

The rest of the lines starting with `//@copy-c` will be generated multiple times when *Code Generator* creates code related to Foo and Bar.

[RTI Issue ID CODEGENII-1679]

### 5.4.4 Publisher Listeners not Functional in Advanced Example for C++98

When generating a C++98 advanced example, the listeners on the publisher side did not work due to an error in their parameters. This problem has been resolved. Now, publisher listeners in the advanced example for C++98 will work correctly.

[RTI Issue ID CODEGENII-1703]

### 5.4.5 Examples Generated with -advanced Option did not Assign QoS Profile to Publishers, Subscribers, or Topics

Because it does not set **is_default_qos** to true, the **-advanced** option for *rtiddsgen* creates entities with the QoS profile specified in **USER_QOS_PROFILES.xml**. *Code Generator*, however, did not apply that profile to Publishers, Subscribers, or Topics. This problem has been resolved. Now the **-advanced** option applies the specified QoS profile to all entities.

[RTI Issue ID CODEGENII-1706]

### 5.4.6 @DDSService Interface Worked only when Defined Last in IDL

You could only define a **@DDSService** interface if it was the last DataType defined in the IDL. This problem has been fixed. Now, you can define more than one **@DDSService** interface in the IDL, and you can define a **@DDSService** interface before other DataTypes in the IDL.

[RTI Issue ID CODEGENII-1708]

### 5.4.7 Unexpected Behavior when allocate_memory was False

When using C++98, *Code Generator* will create the functions **create_data_w_params()**, **create_data()**, and **initialize_data()**; These functions will allow you to create and initialize the Types you had previously generated with *Code Generator*. These functions need a parameter of type DDS_TypeAllocationParams_t, which has an attribute called **allocate_memory**. When calling these functions, **allocate_memory** must be true. If it was false, you may have had unexpected behavior, such as uninitialized members.

This problem has been resolved. Now these functions checks that **allocate_memory** is set to true when calling them. If it is not true, these functions will report an error and **create_data_w_params()** and **create_data()** will return NULL; **initialize_data()** will return DDS_RETCODE_ERROR.

[RTI Issue ID CODEGENII-1740]

# Chapter 6 Known Issues

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

## 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 REQREPLY-37]

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

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

For further details, see the Windows chapter of the *RTI Connext 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++98) 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* 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-666]

## 6.12 64-bit Discriminator Values Greater than (2^31-1) or Smaller than (-2^31) not Supported

Unions with a 64-bit integer discriminator type containing discriminator values that cannot fit in a 32-bit value are not supported when using the following language bindings:

- C
- Traditional C++
- Modern C++
- C#
- Java
- Python
- DynamicData (regardless of the language)

They are also not supported with ContentFilteredTopics, regardless of the language binding.

Using label values greater than 32-bit may lead to receiving samples with invalid content or to filtering samples incorrectly.

[RTI Issue ID CORE-11437]

## 6.13 C# Code Generation for Optional Sequences not Supported

For optional annotation in C#, *Code Generator* will fail and not produce any code.

For example, this IDL will fail for C#:

```
struct MyStruct {
    @optional
    sequence<long, 5> mySquence;
};
```

This issue will be addressed in a future release. For now, the workaround is to use an empty sequence to emulate an unset optional.

For example, for the following IDL:

```
struct Foo {
    sequence<long, 5> mySquence;
};
```

Suppose *Connext* published the following:

```
1.  var sample = new Foo();
2.  sample.mySquence.Add(5);
3.  writer.Write(sample);
4.  sample.mySquence.Clear();
5.  writer.Write(sample);
```

After clearing the sequence in the fourth line, the fifth line will publish an empty sequence, which will emulate an unset optional.

[RTI Issue ID CODEGENII-1503]

## 6.14 Code Generator Performance Degraded After Apache Velocity 2.3 Update

Updating Apache Velocity™ from version 1.7 to 2.3 caused a performance degradation in *Code Generator*. With Apache Velocity 2.3, *Code Generator* 3.1.1 and newer will take up to twice as long for the same IDL as *Code Generator* 3.1.0 or older.

The Apache Velocity update was introduced in *Connext* 6.1.1. Apache Velocity was updated because a vulnerability was found in version 1.7.

[RTI Issue ID CODEGENII-1834]

# 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 message:

> "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.