RTI Code Generator

Release Notes

Version 2.5.0

Copyrights

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 *RTI Core Libraries Release Notes* (available from the RTI Community's Documentation page) 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 listed in the *RTI Core Libraries Platform Notes* except Wind River Linux 3 and 4, and Raspbian Wheezy 7.0.
    - All OS X® platforms listed in the *RTI Core Libraries Platform Notes*
    - All Windows® platforms listed in the *RTI Core Libraries Platform Notes*

Chapter 2 What's New in 2.5.0

2.1 Improved IDL Mapping for Modern C++

Code Generator provides a new, enhanced mapping to modern C++. A new option, **-stl**, combined with **-language C++03** or **-language C++11** enables this mapping.

The new plugin changes the mapping of the following IDL types:

- Unbounded sequences map to **std::vector** (when **-unboundedSupport** is also specified)
- Bounded sequences map to **rti::core::bounded_sequence<T, Bound>**, unless the sequence member has the new annotation **@use_vector** or *rtiddsgen* is run with the new command-line option **-alwaysUseStdVector**, in which case they also map to **std::vector**.
- Strings map to **std::string**.
- Wide strings map to **std::wstring**, wide characters map to **wchar_t**.
- Members with the new annotation **@external** (equivalent to the * pointer notation) map to the new type **dds::core::external<T>**, a type similar to **shared_ptr** (see the API Reference HTML documentation for more details).

For example, given the following IDL type, this is how the generated C++11 type looks with and without **-stl**:

```
struct MyType {
    sequence<long> my_unbounded_seq;
    sequence<long, 10> my_bounded_seq;
    @use_vector sequence<long, 10> my_other_bounded_seq
    string my_str;
    @external long my_external;
};
```

With -stl: rtiddsgen -language C++11 -stl -unboundedSupport MyType.idl

```
class MyType {
    public:
        MyType();
        MyType(const std::vector<int32_t>& my_unbounded_seq,
               const rti::core::bounded_sequence<int32_t,
                10>& my_bounded_seq,
                const std::vector<int32_t>& my_other_bounded_seq,
                const std::string& my_str,
                dds::core::external<int32_t> my_external);

    MyType (MyType&&) = default;
    MyType& operator=(MyType&&) = default;
    MyType& operator=(const MyType&) = default;
    MyType(const MyType&) = default;

    std::vector<int32_t>& my_unbounded_seq() noexcept;
    const std::vector<int32_t>& my_unbounded_seq() const noexcept;

    void my_unbounded_seq(const std::vector<int32_t>& value);
    rti::core::bounded_sequence<int32_t, 10>& my_bounded_seq() noexcept;

    const rti::core::bounded_sequence<int32_t, 10>& my_bounded_seq() const noexcept;
    void my_bounded_seq(const rti::core::bounded_sequence<int32_t, 10>& value);

    std::vector<int32_t>& my_other_bounded_seq() noexcept;
    const std::vector<int32_t>& my_other_bounded_seq() const noexcept;

    void my_other_bounded_seq(const std::vector<int32_t>& value);
    std::string& my_str() noexcept;
    const std::string& my_str() const noexcept;

    void my_str(const std::string& value);
    dds::core::external<int32_t>& my_external() noexcept;
    const dds::core::external<int32_t>& my_external() const noexcept;
    void my_external(dds::core::external<int32_t> value);

    bool operator == (const MyType& other_) const;
    bool operator != (const MyType& other_) const;

    void swap(MyType& other_) noexcept ;
};
```

Without -stl: rtiddsgen -language C++11 -unboundedSupport MyType.idl

```
class MyType {
    public:
        MyType();
        MyType(
            const dds::core::vector<int32_t>& my_unbounded_seq_param,
            const dds::core::vector<int32_t>& my_bounded_seq_param,
            const dds::core::vector<int32_t>& my_other_bounded_seq_param,
            const dds::core::string& my_str_param,
            int32_t * my_external_param);

        MyType (MyType&&) = default;
        MyType& operator=(MyType&&) = default;
        MyType& operator=(const MyType&) = default;
        MyType(const MyType&) = default;

        dds::core::vector<int32_t>& my_unbounded_seq() noexcept;
        const dds::core::vector<int32_t>& my_unbounded_seq() const noexcept;
        void my_unbounded_seq(const dds::core::vector<int32_t>& value);

        dds::core::vector<int32_t>& my_bounded_seq() noexcept;
        const dds::core::vector<int32_t>& my_bounded_seq() const noexcept;
        void my_bounded_seq(const dds::core::vector<int32_t>& value);

        dds::core::vector<int32_t>& my_other_bounded_seq() noexcept;
        const dds::core::vector<int32_t>& my_other_bounded_seq() const noexcept;
        void my_other_bounded_seq(const dds::core::vector<int32_t>& value);

        dds::core::string& my_str() noexcept;
        const dds::core::string& my_str() const noexcept;
        void my_str(const dds::core::string& value);

        int32_t * my_external() const noexcept;
        void my_external(int32_t * value);

        bool operator == (const MyType& other_) const;
        bool operator != (const MyType& other_) const;

        void swap(MyType& other_) noexcept;
};
```

2.2 Compliance with Extensible Types Specification

2.2.1 Support for Built-in Annotations

This release adds support for the following new Extensible Types (XTypes) builtin annotations:

- @autoid
- @hashid
- @value
- @external
- @nested (that is the opposite of the existing annotation topLevel)

In addition, these new names for existing ones are also supported :

- @optional
- @key
- @extensibility(APPENDABLE) or @appendable
- @extensibility(MUTABLE) or @mutable

- @extensibility(FINAL) or @final
- @id
- @nested

All these notations, except autoid, are also supported in XML and XSD when applied to modules.

The following builtin annotations can be defined in IDL, although they will be ignored by the middleware (i.e., they will not be part of the typeobject) : bitset, verbatim, must_understand, min, max, range, bit_bound, default, default_literal, non_serialized, oneway, position, try_construct

Custom annotations can also be defined in IDL, although they are ignored by the middleware (they will not be part of the typeobject).

2.2.2 Support for Prefix Annotations in IDL

This release adds support for prefix annotations in IDL. For example:

```
@id(17) long my_member
```

The comment-based notation is also supported in the standard format and RTI extension format.

Standard format:

```
long id; //@ID (0)
```

RTI extension format:

```
long id; //@ID 0
```

The prefix annotation will be used when converting an XML or XSD file into IDL using the command line option **-convertToIDL**.

2.2.3 Support for Negative Values in Enumerators

This version of Code Generator allows setting negative values to enumerators.

2.2.4 Support for Empty Structures in IDL File

Code Generator threw an error and did not generate code if the IDL contained an empty structure. This release adds support for an empty structure in IDL and will generate code for it.

2.3 Platforms

2.3.1 New Platforms

This release adds support for platforms on the following operating systems:

- OS X® 10.12
- Red Hat® Enterprise Linux® 6.8
- Ubuntu® 16.04 LTS
- VxWorks® 6.9.4 (PPC), VxWorks 7.0 (x64)
- Windows® Server 2016

For details on these platforms, see the *RTI Core Libraries Platform Notes.*

2.3.2 Platforms on Legacy Operating Systems

The following legacy operating systems have reached end-of-life from their corresponding vendors. Please contact RTI Support or your account manager if you require version 5.3 to run on these platforms:

- CentOS™ 5.x
- Red Hat Enterprise Linux 5.x
- SUSE® 11
- VxWorks 6.3, 6.4, 6.6, 6.7
- Wind River® Linux 4

2.3.3 Removed Platforms

Platforms on the following operating systems are no longer supported:

- AIX™ 5.3
- Linux:
  - Linux platforms on PowerPC CPUs (Freescale, Yellow Dog Linux 4, Wind River Linux 3)
  - Red Hat Enterprise Linux 4.0
  - Red Hat Enterprise Linux 5.2 with Real-Time Extensions
- LynxOS® 4.0 on x86 CPU
- OS X 10.8
- Solaris™ 2.9
- VxWorks 5.5, 6.5, VxWorks 653 2.3 on simpc CPU
- Windows Vista®, Windows XP Pro, Windows 2003

2.3.4 New Platforms for Running as Native Application (rtiddsgen_server)

As a native application, Code Generator is supported on select platforms by using the script *rtiddsgen_server*. This script is now also available for these platforms:

- OS X 10.10 and 10.12
- Windows 7 (x86), Windows Server 2008 R2 (x64) with VS 2010

OS X 10.8 platforms are no longer supported.

2.4 Traditional C++ API

2.4.1 Added Default Constructor, Copy Constructor, Copy Assignment Operator, and Destructor to Generated Type–Traditional C++ Only

This release allows you to generate a default constructor, copy constructor, copy assignment operator, and destructor for the generated types in C++ when the new command-line option **-constructor** is used.

For example, the IDL type:

```
struct MyType {
    string member_1;
    sequence<long> member_2;
};
```

Will result in the following C++ type:

```
class MyType
{
    DDS_Char *   member_1;
    DDS_LongSeq  member_2;

    MyType();
    MyType(const MyType& that);
    ~MyType();
    MyType& operator=(const MyType& that);
};
```

Notice that the destructor is not virtual for performance reasons when using ContentFilterTopics. If you want to generate virtual destructors, use the new option **-virtualDestructor**, which can be used in combination with **-constructor**.

When using the **-constructor** option, the generated MyTypeTypeSupport class will no longer implement the following methods:

- **MyTypeTypeSupport::create_data()**
- **MyTypeTypeSupport::create_data_ex()**

- **MyTypeTypeSupport::initalize_data()**
- **MyTypeTypeSupport::initalize_data_ex()**
- **MyTypeTypeSupport::delete_data()**
- **MyTypeTypeSupport::create_data_ex()**
- **MyTypeTypeSupport::finalize_data()**
- **MyTypeTypeSupport::finalize_data_ex()**
- **MyTypeTypeSupport::copy_data()**

2.4.2 Support for Managed Strings in C++ Code Generation–Traditional C++ Only

In previous releases, Code Generator mapped an IDL **string** to a '**char ***' in the traditional C++ API. This release provides a new command-line option for *rtiddsgen*, **-useStdString**, which changes the mapping from '**char ***' to **std::string**.

In the modern C++ API, you can use the new option **-stl** to generate **std::string**.

2.4.3 Simplified Memory Management for Optional Members–Traditional C++ API Only

In previous releases, users were advised to have their applications allocate and release optional members using the following functions:

- **DDS_Heap_malloc()**
- **DDS_Heap_free()**

The type-support also used these, for example, in **FooTypeSupport::create_data()** or **FooTypeSupport::delete_data()**.

Besides being less intuitive than **new** and **delete**, these functions posed a problem allocating optional sequences. Sequences have constructors, so **DDS_Heap_malloc** simply reserved the memory but didn't initialize the object. Using new would be inconsistent with the call to **DDS_Heap_free()** from **FooTypeSupport::delete_data()**. A workaround that matched memory allocation and deallocation functions while still invoking the constructor was to pass memory allocated with **DDS_Heap_malloc()** to operator placement new.

This problem has been resolved. Now all optional members should be created and destroyed with **new** and **delete**.

(Note: In the Modern C++ API, the type dds::core::optional<T> automates memory management.)

2.4.4 New Command-Line Option to Allocate Optional Members using malloc–Traditional C++ Only

This version of Code Generator has simplified the memory management for optional members in C++. To have backward-compatibility when allocating optional members, use the new command-line option, **-allocateWithMallo**c.

2.5 Support for XML Schemas (XSD)

This release introduces support for input files in XSD format. For more information about the supported mappings, see the *RTI Connext DDS Core Libraries User's Manual*, Section 3.5.

This release also introduces the option to transform IDL files into XSD files: **-convertToXsd**. Input XSD files are validated in order to check that they are well-formed.

There is a new option, **-disableXSDValidation**, that can be used to avoid that validation. However, using this option is not recommended in general, as Code Generator may receive invalid inputs.

2.6 Support for Deserializing Union with Unknown Discriminator Value

By default, a *DataReader* cannot receive an unknown union discriminator value from a *DataWriter*. If this situation does occur, the sample containing the unknown discriminator fails to be deserialized. For example, the following two types are assignable:

Publisher Type:

```
union MyUnion switch(long) {
    case 0:
        long m1;
    case 1:
        short m2;
    case 2:
        double m3;
}; //@Extensibility MUTABLE_EXTENSIBILITY
```

Subscriber Type:

```
union MyUnion switch(long) {
    case 0:
        long m1;
    case 1:
        short m2;
}; //@Extensibility MUTABLE_EXTENSIBILITY
```

However, if the *DataWriter* sends a union with the discriminator set to 2, the *DataReader* will not be able to deserialize the sample.

This release adds a new property, **dds.sample_assignability.accept_unknown_union_discriminator**, which allows samples containing an unknown discriminator to be successfully deserialized to the default discriminator value. The default discriminator value is defined as the default element if one is specified, otherwise the lowest value associated with any discriminator value. The member identified by the default discriminator is also initialized to its default value.

You can set **dds.sample_assignability.accept_unknown_union_discriminator** as part of the Property QoS for either the *DomainParticipant* or the *DataReader*. If it is set in both the *DomainParticipant* and *DataReader*, the value in the *DataReader's* QoS will be applied.

This functionality is supported both in generated code as well as when using the DynamicData API.

2.7 Support for Deserializing Enum Member with Unknown Enumerator

By default, a *DataReader* cannot receive an unknown enumeration value from a *DataWriter*. If this situation does occur, the sample containing the unknown enumerator fails to be deserialized. For example, the following two types are assignable:

Publisher Type:

```
enum MyEnum {
    ONE = 1,
    TWO = 2,
    THREE = 3
};
struct MyType {
    MyEnum m1;
};
```

Subscriber Type:

```
enum MyEnum {
    ONE = 1,
    TWO = 2
};
struct MyType {
    MyEnum m1;
};
```

However, if the *DataWriter* sends m1 = THREE, the *DataReader* cannot deserialize the sample.

This release adds a new property, **dds.sample_assignability.accept_unknown_enum_value**, which allows samples containing an unknown enumerator to be successfully deserialized to the default enumeration value. The default enumeration value is defined as the first declared member of the enumeration.

You can set **dds.sample_assignability.accept_unknown_enum_value** as part of the Property QoS for either the *DomainParticipant* or the *DataReader*. If it is set in both the *DomainParticipant* and *DataReader*, the value in the *DataReader's* QoS will be applied.

This functionality is supported both in generated code as well as when using the DynamicData API.

2.8 New TypeSupport Operations to Transform Sample into String Representation

This release provides a new TypeSupport operation that provides a string representation of a sample.

This feature is supported in C, the Traditional C++ API, the Modern C++ API, Java, .NET, the builtin-types, and DynamicData.

C:

```
#include "FooSupport.h"
FooTypeSupport_data_to_string(...)
DDS_DynamicData_to_string(...)
```

Traditional C++:

```
#include "FooSupport.h"
FooTypeSupport::data_to_string(...)
DDS_DynamicData::to_string(...)
```

Modern C++:

```
rti::topic::to_string(...)
```

Java:

```
FooTypeSupport.get_instance().data_to_string(...)
DynamicData::to_string(...)
```

C++/CLI:

```
DDS::FooTypeSupport::data_to_string(...)
DynamicData::to_string(...)
```

C#:

```
using DDS;
FooTypeSupport.data_to_string(...)
DynamicData::to_string(...)
```

2.9 Support for Covariance on FooSeq.get() Method

In previous versions of Code Generator, the **FooSeq.get()** method returned an Object, so a cast was needed to assign the return value to the corresponding type, as in this example:

```
Foo foo = (Foo) fooSeq.get(0);
```

This version of Code Generator supports covariance in the **get()** method. It will return the type of the elements in the sequence, so the cast is no longer needed.

```
Foo foo =  fooSeq.get(0);
```

2.10 Code Generator now Searches Extra Location for Included Files

In previous releases, when a file included more files using the **#include** directive, Code Generator looked for the other files in these directories, in this order:

1. The working directory.
2. The directories specified by the user using the **-I** command-line option.

Now Code Generator will also look here (after the first two locations above):

3. The directory of the input file.

2.11 Faster Code Generation with IDLs Containing Modules in Some Cases

This release enhances the name lookup algorithm used by Code Generator. Previously Code Generator may have taken a long time to generate code from an IDL with multiple nested modules containing references to identifiers without their fully qualified name. The processing time taken by Code Generator in these cases has been reduced with this enhancement.

2.12 Improved Performance of Serialize Method for Mutable Types or Types with Optional Members in Java and C#

In this version of Code Generator, the generated code in Java or C# for the serialize method for mutable types, or types containing optional members, does not need to call **get_serialized_sample_max_size()** on each execution. Consequently, the performance of the serialize method in this case has improved.

2.13 Support for '-typeSequenceSuffix' without Using CORBA Compatibility Kit

The option **-typeSequenceSuffix <Suffix>** is now available even if you do not have the RTI CORBA Compatibility Kit. It can be used with these languages: C, C+, C#, C+/CLI, and Java.

This option assigns a suffix to the name of the implicit sequence defined for IDL types. By default, the suffix is '**Seq**'. For example, given the type '**Foo**,' the name of the sequence would be '**FooSeq**.' If the option **-typeSequenceSuffix <Suffix>** is used, the name of the sequence would be '**Foo<Suffix>**.'

2.14 Improved Error Messages Regarding Enumerator Deserialization

Code Generator now includes new messages when there is an error while deserializing an enumerator.

If the subscriber receives an enumerator value that it does not understand, it will show a message like the one in the following example, indicating the offended value (5), the corresponding enumeration (E1) and the sent type that contained the enumeration (Test).

```
E1Plugin_deserialize_sample:received invalid enumerator value 5
for enumeration E1
TestPlugin_deserialize:Unassignable sample of type Test
PRESPsReaderQueue_storeSampleData:!deserialize
```

Note: In this version of Code Generator, due to the fix of CODEGENII-211, sending invalid enumeration values will not be possible. However, you may see the above error message when a publisher using a previous version of Code Generator communicates with subscribers using this version.

2.15 DataWriter and DataReader Ada Body Types no Longer Generated

This release re-implements code generation for user-created *DataWriters* and *DataReaders* using Ada generics. As a result, a separate Ada body file (*.adb) for each Typed *DataWriter* and Typed *DataReader* is no longer needed and RTI Code Generator will no longer generate a *DataWriter*/*DataReader* Ada body for each type. Two additional generic *DataWriter*/*DataReader* Ada body files will be included in the shipped include headers.

2.16 Option for Compatibility with 5.2.6, 5.2.5, and 5.2 when Using Keyed Mutable Types Inheriting from Another Type with Keyed Fields (-use526Keyhash)

Starting with Connext DDS 5.2.7 and 5.3.0, the Keyhash calculation for a keyed mutable type inheriting from another type containing keyed fields in C, C++ and .NET has changed in order to fix a language interoperability issue. (This is related to RTI Issue ID CODEGENII-693).

If you need a C, C++ or .NET application to communicate with a 5.2.6/5.2.5/5.2.2 C, C++ or .NET applications use the new **-use526Keyhash** flag when running *rtiddsgen* to generate the code for your application.

2.17 Option for Compatibility with 5.2.3 and Prior GARs when using Keyed Mutable Types (-use52Keyhash)

Starting with Connext DDS 5.2.2, the Keyhash calculation for a keyed mutable type in Java and .NET changed in order to fix a language interoperability issue. (This is related to RTI Issue ID CODEGENII-501).

To be backward compatibility with 5.2.3 and prior GAR releases, *rtiddsgen* provided a flag called **-use52JavaKeyhash** that could be used with versions 5.2.2, 5.2.5, and 5.2.6. That flag has been deprecated and replaced with **-use52Keyhash** starting with Connext DDS 5.2.7 and 5.3.0.

Chapter 3 What's Fixed in 2.5.0

3.1 Error in C/C++ when Two Members of Different Enumerations had Same Name

The generated code in C/C++ for IDL containing enumerations with members that have the same name would not compile. For example, consider this IDL:

```
module a {
    module b {
        enum Foo {
            GREEN, RED
        }
    };
};
module c {
    module d {
        enum Bar {
            GREEN, YELLOW
        };
    };
};
```

The generated code for the above IDL was:

```
typedef enum c_d_Foo
{
    GREEN ,
    RED
} c_d_Foo;
typedef enum c_d_Bar
{
    GREEN,
    YELLOW
} c_d_Bar;
```

This caused an error similar to this when trying to compile:

```
test.h:82: error: redeclaration of enumerator 'GREEN'
test.h:25: error: previous definition of 'GREEN' was here
```

This release introduce a new command-line option for RTI Code Generator, **-qualifiedEnumerators**, which allows you to generate fully qualified enumerator names, thus avoiding conflicting names in C/C++.

For example, consider this IDL:

```
module myModule{
    enum Color2 {
        GREEN,
        RED
    };
    union MyUnion switch (Color2){
        case GREEN:
            long m1;
        case RED:
            long m2;
    };
};
```

The following shows the generated code for this IDL:

Language	Without -qualifiedEnumberator	With -qualifiedEnumberator

C	`typedef enum myModule_Color2`
	`{`
	`  GREEN,`
	`  RED`
	`} myModule_Color2;`
	
	`typedef struct myModule_MyUnion`
	`{`
	`  myModule_Color2 _d;`
	`  struct myModule_MyUnion_u`
	`  {`
	`    DDS_Long m1;`
	`    DDS_Long m2;`
	`  }_u;`
	`} myModule_MyUnion;`
		`typedef enum myModule_Color2`
		`{`
		`  myModule_Color2_GREEN,`
		`  myModule_Color2_RED`
		`} myModule_Color2;`
		
		`typedef struct myModule_MyUnion`
		`{`
		`  myModule_Color2 _d;`
		`  struct myModule_MyUnion_u`
		`  {`
		`    DDS_Long m1;`
		`    DDS_Long m2;`
		`  }_u;`
		`} myModule_MyUnion;`

Language	Without -qualifiedEnumberator	With -qualifiedEnumberator

C++	`typedef enum myModule_Color2`
	`{`
	`  GREEN,`
	`  RED`
	`} myModule_Color2;`
	
	`typedef struct myModule_MyUnion`
	`{`
	`  typedef struct myModule_MyUnionSeq Seq;`
	`  myModule_Color2 _d;`
	`  struct myModule_MyUnion_u`
	`  {`
	`    DDS_Long m1;`
	`    DDS_Long m2;`
	`  }_u;`
	`} myModule_MyUnion;`
		`typedef enum myModule_Color2`
		`{`
		`  myModule_Color2_GREEN,`
		`  myModule_Color2_RED`
		`} myModule_Color2;`
		
		`typedef struct myModule_MyUnion`
		`{`
		`  typedef struct myModule_MyUnionSeq Seq;`
		`  myModule_Color2 _d;`
		`  struct myModule_MyUnion_u`
		`  {`
		`    DDS_Long m1;`
		`    DDS_Long m2;`
		`  }_u;`
		`} myModule_MyUnion;`

C+_ Namespace	`namespace myModule {`
	`  typedef enum Color2`
	`  {`
	`    GREEN,`
	`    RED`
	`  } Color2;`
	
	`  typedef struct MyUnion {`
	`    typedef struct MyUnionSeq Seq;`
	`    myModule::Color2 _d;`
	`    struct MyUnion_u`
	`    {`
	`      DDS_Long m1;`
	`      DDS_Long m2;`
	`    }_u;`
	`  } MyUnion ;`
	`};`
		`namespace myModule {`
		`  typedef enum Color`
		`  {`
		`    Color_GREEN,`
		`    Color_BLUE`
		`  } Color;`
		
		`  typedef struct MyUnion`
		`  {`
		`    typedef struct MyUnionSeq Seq;`
		`    myModule::Color2 _d;`
		`    struct MyUnion_u`
		`    {`
		`        DDS_Long m1;`
		`        DDS_Long m2;`
		`     }_u;`
		`  } MyUnion;`
		`};`

[RTI Issues ID CODEGENII-196]

3.2 DataWriter Sent Invalid Enumerator Values without Throwing an Error

If you tried to write a sample containing an invalid enumerator value in C/C++, no error was thrown on the publisher side and the sample was incorrectly sent.

For example, consider this IDL:

```
enum E1 {
    a, b , c
};
struct Test {
   E1 myEnum;
};
```

Sending a sample with a value of 5 for the **E1** enumerator would be incorrect.

```
instance->myEnum=(E1)5;
```

This problem has been resolved. Now the sample will not be sent and you will receive an exception. For the above example, the exception would be:

```
E1Plugin_serialize:invalid enum value 5 for enumerator E1
PRESWriterHistoryDriver_initializeSample:!serialize
```

[RTI Issue ID CODEGENII-211]

3.3 Code Generator did not Handle Programming Keywords

In previous versions, Code Generator didn't consider language-specific keywords when generating code. Therefore, the generated code for an element whose name was a keyword didn't compile.

This problem has been resolved by adding a prefix to those keywords' names in the generated code. The prefix will be as follows, depending on the language:

- C: _c_
- C++, C++03, and C++11: _cxx_
- C++/CLI: _cpp_cli
- .NET: _cs_
- Java: _

It should be noted that the typecode names for those members remain unchanged, i.e, they will not have this prefix.

[RTI Issue ID CODEGENII-291]

3.4 Comment-Based Annotation Syntax for IDL didn't Allow ()

The syntax for comment-based IDL annotations (directives) described in the XTypes Specification defines that the arguments of the annotation should be inside parentheses.

```
{ "//@"<annotation_type_name> [ "(" <arguments> ")" ] }*
```

Previous versions of Code Generator did not allow the parentheses and complained for a directive including them. For example:

```
//@ID (15)
```

This problem has been resolved. Now both annotations with and without parentheses are allowed (in order to preserve backward compatibility).

[RTI Issue ID CODEGENII-431]

3.5 Incorrect Data Serialization for Mutable Types or Types with Optional Members

The TypePlugin function **get_serialized_sample_size()** may have returned the wrong value (greater than expected or smaller than expected) when invoked on MUTABLE types or types containing OPTIONAL members. This problem affected all APIs. There were two consequences visible to users:

- Invoking the **serialize_to_cdr_buffer()** method on a buffer whose size was equal to the size returned by the first call to **serialize_to_cdr_buffer()** may have failed.
- Setting the writer property **dds.data_writer.history.memory_manager.fast_pool.pool_buffer_max_size** may have caused the serialization of samples to fail.

This problem has been resolved in the C, C++, Java, and C# APIs. To use the fix, you must regenerate your code using *rtiddsgen*.

Note that a fix for this issue was introduced in 5.1.1.4 version (CODEGEN-796) for C, C++ and .NET but it wasn't complete and could still be inaccurate for nested types.

[RTI Issue ID CODEGENII-479]

3.6 XML Type Representation for Basic Types not Compliant with Extensible Types Specification

When converting a type containing basic types into XML format, Code Generator used to generate an XML type that was not compliant with the latest Extensible Types Specification.

For example, it generated this for an octet member:

```
<member name="myOctet" type="octet" />
```

but it should have generated this:

```
<member name="myOctet" type="byte" />
```

This problem has been resolved. To maintain backward compatibility, RTI Code Generator still supports XML files with the old notation as an input.

[RTI Issue ID CODEGENII-493]

3.7 Interoperability Problem between C/C++ and Java or .NET applications when Using Keyed Mutable Types

In Connext DDS5.2 and below, the instance keyhash for keyed mutable types was calculated differently in C/C++ and Java/.NET languages. As a consequence, the subscribing application may have observed some unexpected behavior related to instances. Specifically, the call to **DataReader::lookup_instance()** may have failed and returned HANDLE NIL even if the instance was received. Because of this failure, *RTI Spreadsheet Add-In for Microsoft® Excel®* may have failed to show the values for this instance.

This problem has been resolved. Release 2.5.0 includes a new command-line option, **-use52CKeyhash**, which can be used to make Java and .NET instance keyhash generation compatible with C, C++, C++03, and C++11. This option is not enabled by default.

[RTI Issue ID CODEGENII-501]

3.8 Null Pointer Exception when Compiling XML with Union with Member Marked as Key

Code Generator may have thrown a Null Pointer Exception when generating code from XML containing a union with a member marked as key. The key directive is only allowed for members of structs and valuetypes. In this release, Code Generator will log messages like the following one to let you know about the problem:

```
ERROR com.rti.ndds.nddsgen.Main Error generating the rawTree. In member
unionMember1: 'key' directive only applies to members of 'struct' and
'valuetype'
ERROR com.rti.ndds.nddsgen.Main Fail: The file couldn't be parsed and
the rawTree wasn't generated.
```

[RTI Issue ID CODEGENII-532]

3.9 Types Containing Arrays of Pointers to Sequences not Properly Initialized

Types that contained arrays of pointers to sequences were not initialized properly. For example:

```
struct MyType {
    sequence<long,2> * myMember[3];
};
```

Calling **TypeSupport::initialize_data()** or **TypeSupport::create_data()** resulted in an array whose sequences were initialized with a maximum equal to the size of the array (3) instead of the maximum size of the sequence (2). This problem has been resolved.

[]RTI Issue ID CODEGENII-552]

3.10 Setting Maximum of a Sequence to Value Larger than IDL Bound

In previous releases, for sequences contained in types declared in IDL, you could set a maximum that exceeded the bound declared in IDL. For example:

```
struct MyType {
    sequence<MyElementType,3> m1;
};
```

Using the above example, you could have set the maximum for the sequence to a value greater than 3. Once the sample was written with a *DataWriter*, you would have seen a serialize error.

Setting the maximum to something greater than the IDL bound is no longer allowed, except for unbounded sequences in which there is no explicit bound.

[RTI Issue ID CODEGENII-553]

3.11 Server Mode did not Support Parallel Executions

Running parallel executions of rtiddsgen_server was not supported. You may have seen "Address already in use" errors like this:

```
INFO com.rti.ndds.nddsgen.Main Done (failures)
ERROR com.rti.ndds.nddsgen.Main I/O error generating code
java.net.BindException: Address already in use
at java.net.PlainSocketImpl.socketBind(Native Method)
at java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:376)
at java.net.ServerSocket.bind(ServerSocket.java:376)
at java.net.ServerSocket.<init>(ServerSocket.java:237)
at java.net.ServerSocket.<init>(ServerSocket.java:128)
at com.rti.ndds.nddsgen.CodeGeneratorServer.<init>(Unknown Source)
at com.rti.ndds.nddsgen.Main.main(Unknown Source)
```

This problem has been resolved. Now in case of parallel executions, one of the programs may show the warning message below, but code generation will continue successfully.

```
WARN com.rti.ndds.nddsgen.CodeGeneratorServer Cannot bind
to port 14662. Another rtiddsgen in server mode or
application is listening this port.
```

Furthermore, server mode was also unable to handle errors from the code generation. As a consequence, it always exited with the successful return code 0. Now it can detect these errors and return with failure code 255 if the code generation is unsuccessful.

[RTI Issue ID CODEGENII-580]

3.12 Unexpected Error when Sending Samples for Some Types Containing Optional Members – Java API Only

These Java methods may have reported the wrong size when the underlying type was EXTENSIBLE or FINAL and contained optional members:

- **FooTypeSupport::get_serialized_sample_size()**
- **FooTypeSupport::get_serialized_sample_min_size()**
- **FooTypeSupport::get_serialized_sample_min_size()**

As a result, you may have seen an unexpected error when sending samples of that type, such as:

```
Exception in thread "main" com.rti.dds.cdr.IllegalCdrStateException:
not enough available space in CDR buffer
   at com.rti.dds.cdr.CdrBuffer.checkSize(Unknown Source)
   at com.rti.dds.cdr.CdrOutputStream.writeByte(Unknown Source)
   at com.rti.dds.cdr.CdrOutputStream.writeChar(Unknown Source)
```

You may have also seen this error when using the **FooTypeSupport::serialize_to_cdr_buffer()** method.

This problem has been resolved.

[RTI Issue ID CODEGENII-595]

3.13 Missing Required Properties in USER_QOS_PROFILES.xml for Unbounded Support in Java

Code Generator 2.3.3 introduced support for unbounded sequences in Java. However, the generated **USER_QOS_PROFILES.xml** file was missing these properties:

- **dds.data_writer.history.memory_manager.java_stream.min_size()**
- **dds.data_writer.history.memory_manager.java_stream.trim_to_size()**
- **dds.data_reader.history.memory_manager.java_stream.min_size()**
- **dds.data_reader.history.memory_manager.java_stream.trim_to_size()**

These properties must be configured in order to run applications with unbounded sequences or strings. This problem has been resolved.

[RTI Issue ID CODEGENII-597]

3.14 Incorrect Data Serialization for Mutable Enums in Java

The generated Java code for the **serialize()** method for a mutable enum type was incorrect. As a result, a Java publisher of a type containing a mutable enum would not communicate properly with a subscriber (of any language), and the subscriber could fail to deserialize the sample, or could deserialize the sample without errors but receive incorrect data.

[RTI Issue ID CODEGENII-600]

3.15 rtiddsgen_server Generated Code for Another Language if Called Multiple Times for Different Languages

Code Generator may have incorrectly generated code for an additional language when using the rtiddsgen_server script more than once for different languages.

For example if you executed:

```
rtiddsgen_server -language Java hello.idl
rtiddsgen_server -language C++ hello.idl
```

The second call generated C++ code, and mistakenly also generated Java code.

This problem has been resolved.

[RTI Issue ID CODEGENII-601]

3.16 Generated Code did not Compile if Type Started with :: and //@resolve-name was False

The generated code for a member whose type started with **::** and had a //**@resolve-name** directive was incorrect. For example, the generated code for **::PrimitiveType** in this type was incorrect:

```
struct ChildStructure{
    ::PrimitiveType m2; //@resolve-name false
}; //@top-level false
```

This problem has been resolved.

[RTI Issue ID CODEGENII-614]

3.17 Preprocessor Output not Redirected to Code Generator Logging System

When running Code Generator with the preprocessor enabled, its error output was redirected to the standard error output and not logged through Code Generator logging messages. As a consequence, when running *rtiddsgen_server*, the preprocessor output could have been hidden in the output and preprocessor errors wouldn't have been shown in the output.

This version of Code Generator redirects preprocessor errors to the logging system, so its output will be shown just like any other Code Generator error.

[RTI Issue ID CODEGENII-617]

3.18 Server Mode did not Handle Relative Paths when Executed from Different Folder

Running several instances of *rtiddsgen_server* from different directories and with relative path arguments may have failed. In this scenario, the program resolved the relative path with respect to the working directory of the first execution. This problem has been resolved.

[RTI Issue ID CODEGENII-618]

3.19 Incorrect Deserialization of Final Types with Inheritance

The generated code for a type with inheritance and final extensibility was incorrect. An example of that type would be **child** in the following IDL:

```
struct base {
    long a;
};//@Extensibility FINAL_EXTENSIBILITY

struct child : base {
    long b;
};//@Extensibility FINAL_EXTENSIBILITY
```

The deserialization code of the **child** structure was missing the deserialization of the **base** struct. Consequently, a subscriber would have deserialized the data incorrectly. In the example above, if a **child** structure was published with values **a** = 17 and **b** = 33, the subscriber showed the value sent in **a** as the value of **b** in the **child** (that is, **a** = 0 and **b** =17). This problem has been resolved.

[RTI Issue ID CODEGENII-624]

3.20 Server Mode Crashed after Socket Exception

When running Code Generator in server mode, *rtiddsgen_server* may have crashed if a socket exception occurred (such as after a 'connection refused' error). This problem has been resolved.

[RTI Issue ID CODEGENII-627]

3.21 from_cdr_buffer() Leaked Memory if Sample's Data Type Contained Optional Sequence of Strings—Modern C++ API Only

A call to **rti::topic::from_cdr_buffer<Foo>(buffer)**, where the IDL type **Foo** contained an optional string of sequences, such as the following, would have leaked memory.

```
struct Foo {
    sequence<string> seq; //@optional
};
```

This problem has been resolved.

[RTI Issue ID CODEGENII-629]

3.22 Invalid XML File when Using Copy Directives that Included Reserved XML Symbols

When converting to an XML File, Code Generator did not replace reserved XML symbols included in copy directive messages. For example:

```
//@copy-c-declaration #if RTI_DDS_VERSION_MAJOR < 4
```

would generate

```
<directive kind="copyCDeclartion">#if RTI_DDS_VERSION_MAJOR < 4</directive>
```

But that will be an invalid XML because it includes the symbol **<**.

This version of Code Generator replaces those reserved symbols with their corresponding ones in XML. The generated code for the previous example is now:

```
<directive kind="copyCDeclaration">#if RTI_DDS_VERSION_MAJOR &lt ; 4</directive>
```

[RTI Issue ID CODEGENII-647]

3.23 Error When Using Flag Names within File Path Options

In previous versions of Code Generator, if the name of a flag was specified as part of a file or directory path, Code Generator may have thrown an error and not generated code. For example,

```
rtiddsgen -language C++ "C:\test-microwaves\test.idl"
```

would have caused this error to be thrown, because the file path contained the name of the "-micro" flag:

```
ERROR velocityEngine ResourceManager : unable to find resource 'micro_c\typeHeader.vm' in any
resource loader.
```

This problem has been resolved.

[RTI Issue ID CODEGENII-651]

3.24 Serialize_to_cdr_buffer() API Used Wrong Endianness on Little-Endian Machines

The generated code for the **serialize_to_cdr_buffer()** API in C, C++, C++03, and C++11 used the wrong endianness to serialize into the CDR buffer on little-endian machines. As a consequence, serializing and deserializing data using this API on little-endian machines, in conjunction with the DynamicData equivalent APIs **DynamicData_to_cdr_buffer()** and **DynamicData_from_cdr_buffer()**, produced incorrect results. This problem has been resolved.

[RTI Issue ID CODEGENII-657]

3.25 Java serialize_to_cdr() Threw "java.lang.OutOfMemoryError" for Keyed Types with Unbounded Members

Using the Java serialize_to_cdr() method to get the size of a serialized sample of a keyed type with unbounded members, caused an error:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

For example:

```
struct KeyedUnboundedType {
    octet key; //@key
    sequence<octet> data;
};
KeyedUnboundedType.get_instance().serialize_to_cdr_buffer(null, 0, instance);
```

This problem, which affected *Code Generator* 2.3.3.14, 2.3.4, 2.3.5 and 2.3.6 , has been resolved.

[RTI Issue ID CODEGENII-680]

3.26 Error Exporting <enum_type>_get_default_value() method in Type DLL (C++/CLI)

The generated method **<enum_type>_get_default_value()** in C++/CLI for an enumeration was not exported properly when creating a type DLL containing the enumeration. Consequently, referencing that method from a different application or DLL generated the following error:

```
error C3861: '<enum_type>_get_default_value': identifier not found
```

This problem has been resolved.

[RTI Issue ID CODEGENII-684]

3.27 Interoperability Problem between Java and C/C++/NET Applications when Using Keyed Mutable Types and Inheritance

The instance keyhash for a keyed mutable type inheriting from another type containing key fields was calculated differently in the Java and C/C++/.NET languages. For example:

```
struct MyBase {
    long m1; //@key
}; //@Extensibility MUTABLE_EXTENSIBILITY

struct MyDerived: MyBase {
    long m2; //@key
}; //@Extensibility MUTABLE_EXTENSIBILITY
```

The right calculation was done in Java.

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

This problem has been resolved. For compatibility purposes, this release introduces a new command-line option, **-use52Keyhash**, which can be used to go back to the keyhash generation used in Connext DDS 5.2.3 and earlier GARs. For backward compatibility with Connext DDS 5.2.6, use **-use526Keyhash** instead.

[RTI Issue ID CODEGENII-693]

3.28 XSD Schema File did not Define Default Value for topLevel

The XSD schema provided to validate XML files didn't contain a default value for the topLevel attribute. This problem has been resolved and the default value is now set to true.

[RTI Issue ID CODEGENII-698]

3.29 Inefficient Serialization and Deserialization of Sequences of Doubles

The serialization and deserialization of sequences of doubles was less efficient than serialization or deserialization of other primitive types. This problem has been resolved.

[RTI Issue ID CODEGENII-701]

3.30 Warnings about Unused Parameters—Modern C++ API Only

Applications that included the Connext DDS Modern C++ API may have seen some warnings about unused parameters, depending on the compiler version and the compilation options. All these warnings have been fixed. Note however that none of these warnings revealed any underlying defect.

[RTI Issue ID CODEGENII-708]

3.31 Possible Segmentation Fault by DataWriter for Topic with Mutable or Optional Members when Using Batching

An application writing samples with a DataWriter with batching enabled may have issued a segmentation fault or printed the following precondition error if the associated Topic had a type marked as mutable or containing optional members:

```
RTICdrStream_serializeParameterLength:!precondition: me
== ((void *)0) || lengthPosition == ((void *)0) || state == ((void *)0)
```

The issue only occurred when the maximum size of the batch was limited using **writer_qos.batch.max_data_bytes**.

This problem has been resolved.

[RTI Issue ID CODEGENII-711]

3.32 Wrong Value Returned from get_serialized_sample_size() for Mutable Types or Types with Optional Members that had Inheritance – Java API Only

This issue only affected patches 5.2.3.x, when x is 14 or higher.

The TypePlugin function **get_serialized_sample_size()** may have returned the wrong value (greater than expected or smaller than expected) when invoked on MUTABLE types or types containing OPTIONAL members that had INHERITANCE. This problem only affected the Java API.

There were two consequences that were user visible:

- Invoking **serialize_to_cdr_buffer()** on a buffer whose size was equal to the size returned by the first call to **serialize_to_cdr_buffer()** may have failed.
- Serializing a sample with a size greater than the one specified in **dds.data_writer.history.memory_manager.java_stream.min_size** may have failed.

This problem has been resolved.

[RTI Issue ID CODEGENII-725]

3.33 Generated C++11 Code for Subscriber on iOS Platforms did not Compile

The generated C++11 code for a subscriber on an iOS platform was wrong and might not have compiled. This problem has been resolved.

[RTI Issue ID CODEGENII-734]

3.34 Parent Member for Ada Generated Type Code was not Aliased

Only the member fields of a generated type were declared as **aliased**, while the parent member was not. This prevented use of the **Access** attribute on the parent type. This problem has been resolved.

[RTI Issue ID CODEGENII-736]

3.35 Code Generator Failed when Running in Parallel with Same Output Folder

When running multiple instances of Code Generator in parallel and with the same output folder, Code Generator may have thrown an exception when creating directories and would not have generated code. This problem has been resolved.

[RTI Issue ID CODEGENII-737]

Chapter 4 Known Issues

4.1 Bug in JDK 1.8.0_121 Affects XSD Files with Several maxOccurs or minOccurs

When generating code for an XSD file in which a member contains several **maxOccurs** or **minOccurs** as seen below, Code Generator might throw a java.lang.OutOfMemoryError: Java heap space error.

```
<xsd:complexType name= "XTypeMutable_needExtendedMemberId_SequenceOflong" >
    < xsd:sequence>
        < xsd:element name= "item" minOccurs = "0"
          maxOccurs= "70000" type ="xsd:int" />
    </ xsd:sequence>
</xsd:complexType >
```

This error occurs during the validation of the XSD file when using jdk/jre version 1.8.0_121. To avoid this error, run *rtiddsgen* with the option **-disableXSDValidation** to skip that validation step.

[RTI Issue ID CODEGENII-489]

4.2 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]

4.3 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 {
    short outer_short;
    struct Inner {
        char inner_char;
        short inner_short;
    } outer_nested_inner;
};
```

XML:

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

[RTI Issue ID CODEGEN-54]

4.4 .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]

4.5 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]

4.6 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]

4.7 Unable to Detect if Optional Member is Inside Aggregated Key Member

Code Generator cannot detect if an optional member is inside an aggregated key member.

[RTI Issue ID CODEGEN-605]

4.8 QNX 6.5.1 on PPC Requires -stl Option to Generate Code for Modern C++

For the QNX 6.5.1 on PPC architecture (armv7aQNX6.5.0SP1qcc_cpp4.4.2): RTI Code Generator (*rtiddsgen*) may generate incorrect C++03 or C++11 code for types that contain boolean members if, in the target platform, sizeof(bool) != 1. The generated code will fail to serialize or deserialize these types.

This problem will not occur if the option **-stl** is used.

[RTI Issue ID CODEGENII-528]

4.9 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 throw 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{
        long 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 a, underscore characters ('_') to the idl module name.

[RTI Issue ID CODEGENII-661]

4.10 Decreased Performance on Content Filtering in Modern C++ STL Type Plugin

The code generator option **-stl** (combined with **-language C++03** or **-language C++11**) generates an improved type plugin that maps types such as strings and sequence to STL types (std::string, std::vector). However, this plugin, unlike the plugin generated without the **-stl** option, requires a different, less-performing algorithm for content filtering, especially for large types. This may impact a Connext DDS application's performance when using ContentFilteredTopics. This issue will be addressed in a future release.

[RTI Issue ID CORE-6652]

Chapter 5 Limitations

5.1 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 {
    long 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]

Chapter 6 Third-Party Licenses

Portions of *RTI* Code Generator were developed using:

- Apache log4j™ from the Apache Software Foundation (http://logging.apache.org/log4j/)
- Apache Velocity™ from the Apache Software Foundation (http://velocity.apache.org/)
- ANTLR v3 (http://www.antlr3.org/)

6.1 Apache Software License Version 2.0

6.2 ANTLR 3 License