RTI Code Generator

Release Notes

Version 3.0.0

Printed in U.S.A. First printing.
February 2019.

Trademarks

Copy and Use Restrictions

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 Connext DDS 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 on x86/x64 CPUs listed in the *RTI Connext DDS Core Libraries Release Notes*, except Wind River® Linux 7.
  - All Windows® platforms listed in the *RTI Connext DDS Core Libraries Release Notes*.
  - For custom supported platforms: RedHawk™ 6.5, Red Hat® Enterprise Linux 5.2.

Chapter 2 Compatibility

For backward compatibility information between 6.0.0 and previous releases, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

Code Generator has been tested with Oracle JRE version 8, which is included in the installation package. No other versions of Java are supported.

Chapter 3 What's New in 3.0.0

3.1 Support for XCDR encoding version 2

This release adds support for the standard XCDR encoding version 2 data representation described in the "Extensible and Dynamic Topic Types for DDS" specification. This encoding version is more efficient in terms of bandwidth than the predecessor XCDR encoding version 1 supported in previous *Connext DDS* releases (and still supported in this release).

Code Generator can generate TypePlugin code that understands both XCDR2 and XCDR encapsulations. To select between XCDR and XCDR2 data representations, you can use the DataRepresentationQosPolicy for *DataReaders* and *DataWriters* (see the *RTI Connext DDS Core Libraries User's Manual*). In addition, the supported encoding versions can be selected on a per type basis using the new annotation **@allowed_data_representation** (see the "Data Representation" chapter of the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*).

3.2 Support for @allowed_data_representation annotation

The **@allowed_data_representation** annotation lets you restrict the data representations that may be used to encode a data object of a specific type. For example:

```
@allowed_data_representation(XCDR2)
struct Position
{
    int32 x;
    int32 y;
};
```

DataWriters and *DataReaders* using the above type can publish and subscribe to only an XCDR2 representation, regardless of the value set in the DataRepresentationQosPolicy. (If the *DataWriter* or *DataReader* in this case sets its DataRepresentationQosPolicy to XCDR, *Connext DDS* will automatically change it to XCDR2 and print a log message indicating this change.)

The @allowed_data_representation value is a bitmask; therefore, it can take multiple values. For example:

```
@allowed_data_representation(XCDR2|XCDR)
struct Position
{
int32 x;
int32 y;
};
```

DataWriters and *DataReaders* using the previous type can publish and subscribe to XCDR or XCDR2 data representations. You can select the specific data representations from within the allowed set by setting the DataRepresentationQosPolicy.

For additional information, see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types* and the *RTI Connext DDS Core Libraries User's Manual*.

3.3 New API to serialize data to CDR buffer with XCDR or XCDR2 data representation

This release includes a new API that allows you to serialize data to a Common Data Representation (CDR) buffer choosing the desired data representation parameter (DDS_AUTO_DATA_REPRESENTATION, DDS_XCDR_DATA_REPRESENTATION, or DDS_XCDR2_DATA_REPRESENTATION):

In C:

```
FooTypeSupport_serialize_data_to_cdr_buffer_ex(
    char *buffer,
    unsigned int *length,
    const Foo *sample,
    DDS_DataRepresentationId_t representation)
```

In Traditional C++:

```
FooTypeSupport::serialize_data_to_cdr_buffer_ex(
    char *buffer,
    unsigned int &length,
    const Foo *sample,
    DDS_DataRepresentationId_t representation)
```

In Java:

```
public long serialize_to_cdr_buffer(
        byte[] buffer,
        long length,
        Foo src,
        short representation);
```

In .NET:

```
FooTypeSupport::serialize_data_to_cdr_buffer(
    array<System::Byte>^ buffer,
```

```
    System::UInt32% length,
    Foo^ a_data,
    System::Int16 representation)
```

In Modern C++ :

```
std::vector<char>& to_cdr_buffer(
    std::vector<char>& buffer,
    const Foo& sample,
    dds::core::policy::DataRepresentationId representation
        = dds::core::policy::DataRepresentation::xcdr());
```

If the representation parameter is not provided, the API will serialize data using DDS_AUTO_DATA_REPRESENTATION. If the type is FlatData, passing in DDS_XCDR_DATA_REPRESENTATION will result in an error because FlatData only supports XCDR2.

3.4 New optimization level for code generation

This release introduces a new optimization level for code generation for C, C++, and Ada languages that can increase the performance of the serialize/deserialize operations significantly in some cases.

This optimization level is enabled by default. It can also be enabled explicitly by using the command line option **-optimization** with value 2.

With optimization level 2, *rtiddsgen* optimizes the serialization/deserialization of structures and valuetypes by using more aggressive techniques, such as inline expansion of nested types or serialization of several consecutive members with a single copy (memcpy).

For example:

```
struct Point {
    long x;
    long y;
};
struct PointArray {
    Point pa[1024];
};
```

In previous versions of *Code Generator*, the serialization of a sample with type PointArray iterated through each one of the elements of the array, serializing each one individually. With optimization level 2, *Code Generator* detects that the memory representation of a PointArray sample is equal to the wire representation and does the serialization with a single memcpy call. The same optimization is applied on deserialization, assuming that the endianness of the serialization buffer matches the endianness of the architecture where the sample is deserialized.

For additional information on this feature, see the *Code Generator User's Manual*.

3.5 Support for new standard IDL fixed-width integer types

This release introduces a new set of standard, fixed-width integer types to improve the readability of IDL files. These types are int16, int32, int64, uint16, uint32, and uint64, which are equivalent to the respective short, long, long long, unsigned short, unsigned long, and unsigned long long classic integer types. For example, the following IDL:

```
struct MyStruct {
   int16 my_16_bit_signed_integer;
   int32 my_32_bit_signed_integer;
   int64 my_64_bit_signed_integer;
   uint16 my_16_bit_unsigned_integer;
   uint32 my_32_bit_unsigned_integer;
   uint64 my_64_bit_unsigned_integer;
};
```

is equivalent to the following:

```
struct MyStruct {
    short my_16_bit_signed_integer;
    long my_32_bit_signed_integer;
    long long my_64_bit_signed_integer;
    unsigned short my_16_bit_unsigned_integer;
    unsigned long my_32_bit_unsigned_integer;
    unsigned long long my_64_bit_unsigned_integer;
}
```

These new types are part of the new Interface Definition Language (IDL) 4.2 specification, which has been recently published by the Object Management Group. The language mapping of the new, fixed-width integers remains the same as that of the equivalent classic integer types.

3.6 Support for @range and @default annotations

This release introduces support for the following annotations:

- **@default** allows you to specify a default value for a primitive, enum, or string member, and it overwrites the default "zero."
- **@default_literal** can be used to select the default enumerator in an enum.
- **@range**, **@min**, and **@max** can be used to restrict the possible values for a primitive member.

For additional information, see the *RTI Connext DDS Core Libraries Getting Started Guide Addendum for Extensible Types*.

3.7 Updated default type mapping when generating code for C++03/C++11

In 2.5.0, a new option, **-stl**, was introduced to change the mapping of some of the IDL types. From this release onward, **-stl** is the default option when generating code for C++03/C++11.

In 3.0.0, a new option, **-legacyPlugin**, combined with **-language C++03** or **-language C++11**, has been introduced to generate code using the old mapping.

For compatibility information related to this change, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

3.8 Type getters and setters are now inline (C++03/C++11)

In previous releases, the field getter and setter functions for a class generated for C++03 or C++11 were declared in the .hpp file and defined in the .cxx. In this release, they are declared and defined inline in the .hpp file.

This change should provide better performance for data-intensive applications.

3.9 Removed support for -notypecode

Code Generator no longer supports the **-notypecode** option. Type code information is always generated, but it is surrounded by

```
#ifndef NDDS_STANDALONE_TYPE
#endif
```

When using standalone types, you already have to add the preprocessor definition NDDS_STANDALONE_TYPE, so now this definition already excludes the type code.

For compatibility information related to this change, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

3.10 Removed support for -use42eAlignment

Code Generator no longer supports the **-use42eAlignment** option.

For compatibility information related to this change, see the *Migration Guide* on the RTI Community Portal (https://community.rti.com/documentation).

Chapter 4 What's Fixed in 3.0.0

4.1 Unable to detect if optional member was inside aggregated key member

Optional members cannot be parts of keys, but previously *Code Generator* did not detect that. It generated code without reporting an error.

This happened in cases like the following, in which the same member is marked as keyed and optional:

```
struct BadType {
    @key
        @optional
        long key_and_optional;
};
```

It also happened when the optional member was inside an aggregated type used as a key:

```
struct NestedType {
    @optional
    long optional_member;
};
struct BadType {
    @key
    NestedType undetected_bad_key;
};
```

Code Generator now reports an error and does not generate code for an invalid IDL containing optional key members.

[RTI Issue IDs CODEGENII-123 and CODEGEN-605]

4.2 Deserialization error in unions without default discriminator when using JacORB 3.x

Although JacORB 3.x was not officially supported in previous releases, if you had tried to use it with a union type without a default discriminator (see type below), the *DataReader* would have

printed deserialization errors and the samples would not have been provided to the application.

Unions with a boolean discriminator and case values for TRUE and FALSE were not affected. Unions with an enum discriminator with a case value for each possible enum value were not affected.

```
union CharUnion switch (char)
{
    case 'B':
        octet octet_mem;
    case 'S':
        short short_mem;
/* There is no default discriminator */
};

struct StructWithUnion {
    CharUnion member_1;
};
```

This problem has been resolved.

[RTI Issue ID CODEGEN-827]

4.3 Linking errors for CCK generated example using ACE-TAO

The compilation of the generated example (using the **-example** flag) for the *RTI Corba Compatibility Kit* (CCK) and ACE-TAO may have failed with linking errors if you did not use the command-line option **-orb** when generating the example code.

For example, the example generated with this command line failed to compile:

```
../scripts/rtiddsgen -corba MyTypeC.h -example ppc7400Lynx5.0.0gcc3.4.3 MyType.idl
```

The example generated with this command line did compile:

```
../scripts/rtiddsgen -corba MyTypeC.h -orb ACE_TAO1.6 -example ppc7400Lynx5.0.0gcc3.4.3
MyType.idl
```

This problem has been fixed. Now the first example will compile.

[RTI Issue ID CODEGEN-834]

4.4 Generated makefile for Java examples for Windows did not work if cygwin was in path

When compiling generated Java code using the generated makefile, you may have seen this error if you had cygwin in your path environment variable:

```
The library nddsjava.dll could not be loaded by Windows.

Make sure that the library is in your Path environment variable.
```

```
Exception in thread "main" java.lang.UnsatisfiedLinkError: no nddsjava in java.library.path

at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1867)

at java.lang.Runtime.loadLibrary0(Runtime.java:870)

at java.lang.System.loadLibrary(System.java:1122)
```

The root cause was that the makefiles were setting the PATH variable instead of the Path one, so the RTI Connext DDS libraries couldn't be found. This issue has been fixed.

[RTI Issue ID CODEGENII-295]

4.5 Improved @resolve-name conversion to XML when applied to struct or union

In previous versions, when IDL containing an @resolve-name directive applied to a struct or union was converted to XML or XSD, all the members of the struct had the resolve name information.

Example IDL:

```
struct MyStruct {
MyStruct2 m1;
}; //@resolve-name false
```

generated

```
<struct name= "MyStruct" resolveName="false">
<member name="m1" id="0" type="nonBasic"  nonBasicTypeName= "MyStruct2" resolveName="false" />
</struct>
```

For the above IDL, there was no way of differentiating in the output XML if the @resolve-name tag was originally applied to the member or to the struct.

In this version of *Code Generator*, the conversion reflects where the @resolve-name tag was set. The generated XML for the above example is now:

```
<struct name= "MyStruct" resolveName="false">
<member name="m1" id="0" type="nonBasic"  nonBasicTypeName= "MyStruct2"/>
</struct>
```

[RTI Issue ID CODEGENII-354]

4.6 Improved error messages when sequence of sequences used in IDL

Defining a sequence of sequences in IDL is currently not supported by *Code Generator*. For example, this is not supported:

```
struct Hello {
sequence < sequence <long, 4> , 5> m1;
};
```

Previous versions of *Code Generator* reported a Null pointer exception if one of these sequences was found, without explaining the problem. This version of *Code Generator* shows a message indicating the problem and how to fix it, like this:

```
ERROR com.rti.ndds.nddsgen.Main codegenii391.idl line 2:15 Sequence of sequences are not
supported. Please use an alias/typedef instead.
```

[RTI Issue ID CODEGENII-391]

4.7 Regenerated code may not have compiled if -replace option was not used

Regeneration of code for an IDL file for which you previously generated code may not have compiled if you did not use the **-replace** command-line option to regenerate the code. In these cases, you would have seen errors like these:

```
MyTypePlugin.cxx:169:5: error: use of undeclared identifier '$'
    ${member.elementPrintMethod}(
    ^
MyTypePlugin.cxx:331:14: error: use of undeclared identifier '$'
        if (!${member.elementSerializeMethod}(
```

These errors occurred only when you:

1. generated code for an IDL file
2. deleted one of the generated files (for example <Type>Plugin.cxx), but not the others
3. regenerated the code without using -replace

This problem has been resolved.

[RTI Issue ID CODEGENII-468]

4.8 Code Generator did not accept constants as enumerator values

Code Generator did not allow assigning a constant as a value for an enumerator, as shown in the following example:

```
const long MYCONST=1;
enum MyEnum {
  MYENUM1 = MYCONST,
   MYENUM1 =2
};
```

This problem has been resolved.

[RTI Issue ID CODEGENII-550]

4.9 IDL containing struct or field name called "position" might not have compiled

The generated code for an IDL containing a struct or a field name called "position" might not have compiled due to a collision with a local variable. This issue has been fixed by changing the name of the local variables from "position" to "rti_position."

[RTI Issue ID CODEGENII-559]

4.10 Generated code did not compile in C# if it contained reserved keywords as type names

In previous releases, if a primitive type name, such as UInt32 in C#, was used as a type name, the generated code did not compile. For example:

```
struct UInt32
{ unsigned short data; }
;
```

In this release, primitive type names have been added as part of the keywords list for the C# language and the prefix _cs_ is used to escape the keyword. So for the above example, the struct name in the generated code will be _cs_UInt32.

[RTI Issue ID CODEGENII-565]

4.11 Generated code for sequences in .Net reported a signed/unsigned mismatch warning

When compiling the generated code for an IDL containing sequences in .Net, you may have seen the following warning:

```
Warning C4018: '<' : signed/unsigned mismatch in .Net
```

The signed/unsigned issue has been fixed, and the warning will no longer appear.

[RTI Issue ID CODEGENII-645]

4.12 Generated examples in Ada did not mention the right logging packages

Publisher and subscriber code generated for Ada examples contained commented-out lines to increase logging verbosity and a comment instructing you to un-comment those lines to change the verbosity level. Code in the commented-out lines used the wrong packages and would fail to compile after removing the comment markers. This problem has been resolved.

[RTI Issue ID CODEGENII-670]

4.13 Invalid behavior in Code Generator when mixing extensibility kinds when using inheritance

Using mixed extensibility kinds when using inheritance is not supported; however, in the previous release, *Code Generator* mistakenly proceeded to generate code in this scenario. This resulted in code that failed to serialize the data. This problem has been resolved. Now *Code Generator* will properly fail if there are mixed extensibility kinds when using inheritance.

[RTI Issue ID CODEGENII-691]

4.14 Error converting to XML for union type with //@resolve-name false directive

When *Code Generator* converted a union type that had an //@resolve-name false directive from IDL to XML, it also applied the directive to the discriminator of the type. For example:

```
<discriminator type="boolean" resolveName="false"/>
```

That is not supported and if the resulting XML was used to generate code, it would have produced a parsing error. This problem has been resolved.

[RTI Issue ID CODEGENII-699]

4.15 Code Generator failed to generate code when the input file contained a native type

Code Generator failed to generate code when the input file contained a native type. For example, when generating code for the following IDL:

```
native Foo;
```

Code Generator produced the following error and did not generate code:

```
ERROR com.rti.ndds.nddsgen.Main Fail: org.antlr.runtime.tree.RewriteEmptyStreamException: rule
type_dcl
```

This problem has been resolved. Now *Code Generator* ignores the native declaration. *Code Generator* now shows the following warning when the input file contains a native type:

```
WARN com.rti.ndds.nddsgen.antlr.auto.IdlParser ... line 1 native Foo will be ignored
```

[RTI Issue ID CODEGENII-762]

4.16 get_serialized_key_max_size() in Java returned bigger value for unkeyed mutable types

The **get_serialized_key_max_size()** method in Java returned a bigger value than it should have for unkeyed mutable types. It was adding twice the sentinel size. This issue has been resolved.

[RTI Issue ID CODEGENII-774]

4.17 get_serialized_max_size and get_serialized_min_size methods returned bigger size for mutable unions in Java

The **get_serialized_max_size** and **get_serialized_min_size** methods returned value sizes that were bigger than they should have been for mutable unions in Java. This issue has been resolved.

[RTI Issue ID CODEGENII-775]

4.18 Loading a generated Visual Studio solution reported an error and disabled auto-completion

The generated Visual Studio project contained an invalid separator comma (,) instead of the standard Windows separator semi-colon (;) in the preprocessor definitions section for the Static Debug configuration:

```
<PreprocessorDefinitions>WIN32;RTI_WIN32;_DEBUG;_CONSOLE,RTI_STATIC;%
(PreprocessorDefinitions)</PreprocessorDefinitions>
```

As a result, loading the project reported an error and disabled the auto-completion:

```
command-line error: invalid macro definition: _CONSOLE,RTI_STATIC
```

This problem has been resolved. Now the generated Visual Studio contains the valid separator (;):

```
<PreprocessorDefinitions>WIN32;RTI_WIN32;_DEBUG;_CONSOLE;RTI_STATIC;%
(PreprocessorDefinitions)</PreprocessorDefinitions>
```

[RTI Issue ID CODEGENII-782]

4.19 get_serialized_min_size() and get_serialized_key_max_size() returned bigger value for type containing array of complex types in Java

The **get_serialized_min_size()** and **get_serialized_key_max_size()** methods returned bigger values than they should have for a type containing arrays of complex types in Java. This issue has been resolved.

[RTI Issue ID CODEGENII-784]

4.20 Get sample_size, max_size, and min_size methods returned bigger value for mutable enums

The **sample_size**, **max_size**, and **min_size** methods in C/C++ and Java returned a bigger value than they should have for mutable enums. The serialization of mutable enums should not contain a sentinel, but these methods were adding the sentinel size, returning a bigger value than the real one. This issue has been resolved.

[RTI Issue ID CODEGENII-785]

4.21 Different output directory for C# applications generated with Code Generator

In releases 2.5.0.7 and 2.5.0.8, when generating code for C# using the **-example** flag for VS2015 or VS2017, the configuration of the generated Visual Studio project was different than in previous releases. When the project was compiled in 2.5.0.7 and 2.5.0.8, the executable was placed into a different directory: into **bin/[x64]/Release-<VSNumber>** instead of the usual one, **bin//[x64]/Release-<VSVersion>**. (VSNumber=14 for VSVersion=VS2015, and VSNumber=15 for VSVersion=VS2017.)

This release fixes this issue. The output path is now **bin//[x64]/Release-<VSVersion>**.

[RTI Issue ID CODEGENII-820]

4.22 Code Generator server preserved flags from previous IDL code generation

Running *Code Generator* in server mode using the **rtiddsgen_server** script could have incorrectly generated code due to the use of options from previous executions. This problem has been resolved.

[RTI Issue ID CODEGENII-826]

4.23 Return values of TypeSupport and sequence functions were not used

The generated code for types containing sequences in C, C+, and modern C++ contained calls to functions whose return values where not checked. For instance, the following code was generated:

```
Foo& FooSeq::set_at(DDS_Long i, const Foo& val) {
    Foo_copy(TSeq_get_reference(this, i), &val);
    return *FooSeq_get_reference(this, i);
}
```

Some static analysis tools detected that the return value was not checked, reporting this issue as a warning. Although the missing return value check was harmless in this context, *Code Generator* 's generated code now checks for the return value.

[RTI Issue ID CODEGENII-827 and CORE-8945]

4.24 Lines added using the //@copy-java-declaration-begin directive were incorrectly copied in clear() method

When the @//copy-java-declaration-begin directive was used to add lines to the type declaration in the generated code for Java, those lines were also copied in the **clear()** method. In that case, the generated code

might have not compiled. This problem has been resolved.

[RTI Issue ID CODEGENII-830]

4.25 Error deserializing samples containing mutable/optional members in Java

A Java *DataReader* may have failed to deserialize a sample when these two conditions were met:

1. The top-level topic type has a maximum serialized size greater than 32767, and smaller than or equal to 65535.
2. The actual serialized size of a mutable/optional member within the sample (it could be a member of a nested type) has a serialized length greater than 32767, and smaller than or equal to 65535.

```
com.rti.dds.cdr.IllegalCdrStateException: not enough available space in CDR buffer
```

For example:

```
@mutable
struct MyType {
    string<128> m1;
    sequence<string<128>,255> m2;
};
```

A sample from the above type, where m2 is populated with 255 sequences of 128 characters, would fail to deserialize in Java because the serialized length of m2 is 34684.

This problem has been fixed.

[RTI Issue ID CODEGENII-831]

4.26 Traditional C++ code could not be compiled with -fno-exceptions

Starting in 5.3.0, the generated code for traditional C++ could not be compiled with the flag **-fno-exceptions**, producing an error similar to this one:

```
In file included from Hello.cxx:215:0:
rti_connext_dds-5.3.0/include/ndds/dds_c/generic/dds_c_sequence_TSeq.gen: In function ‘DDS_
Boolean HelloSeq_set_maximum(HelloSeq*, DDS_Long)’:
rti_connext_dds-5.3.0/include/ndds/dds_c/generic/dds_c_sequence_TSeq.gen:548:32: error:
exception handling disabled, use -fexceptions to enable
} catch (std::bad_alloc&) {
```

This issue has been resolved: the code will not report exceptions, provided that you generate code with the **-allocateWithMalloc** flag. This flag disables the generation of default constructors/destructors and allocates the optional members using DDS_Heap_malloc.

[RTI Issue ID CODEGENII-839]

4.27 Error using @bit_bound(32) annotation

Currently, *Connext DDS* supports enumerators with a bit_bound of "32", which is the default value; however, when explicitly setting the annotation "@bit_bound(32)", *Code Generator* printed the following error message:

```
ERROR com.rti.ndds.nddsgen.Main Fail:  java.lang.ClassCastException:
com.rti.ndds.nddsgen.antlr.annotation.BitBoundAnnotation cannot be cast to
com.rti.ndds.nddsgen.antlr.annotation.ExtensibilityAnnotation
```

This problem has been resolved: the annotation "@bit_bound(32)" can now be used in the type.

[RTI Issue ID CODEGENII-841]

4.28 Code Generator failed to generate code when using octets as union discriminator

Code Generator failed to generate code when using octets as a union discriminator, the usage of which is supported by the Extensible Types specification (https://www.omg.org/spec/DDS-XTypes).

For example, when generating code for the following IDL:

```
module MainType {
        union test switch (octet){
                case 'a': long M1;
        };
};
```

Code Generator produced the following errors and did not generate code:

```
ERROR com.rti.ndds.nddsgen.antlr.auto.IdlParser ... line 5:20 no viable alternative at input
'octet' in union
ERROR com.rti.ndds.nddsgen.Main Fail:  java.lang.Exception: The file couldn't be parsed and the
rawTree wasn't generated
```

This problem has been resolved. Now a union of octets is accepted.

[RTI Issue ID CODEGENII-847]

4.29 Generated code in Java for a type containing a keyed array of sequences did not compile

In versions 2.5.0.7, 2.5.0.8, and 2.5.2 of *Code Generator*, the Java-generated code for a keyed array of sequences, such as the following, was incorrect and did not compile:

```
sequence<long long,10> myLongLongSeqArr[2]; //@key
```

This problem has been resolved.

[RTI Issue ID CODEGENII-849]

4.30 Incorrect mapping of IDL "const string" to C++

According to the Object Management Group (OMG) specification "C++ Language Mapping," the mapping of "const string" from IDL to C++ should be:

```
// IDL
const string name = "testing";
// C++
static const char *const name = "testing";
```

Previous versions of *Code Generator*, however, mapped "const string" to the following:

```
// C++
static const char * name = "testing";
```

Since the second "const" modifier was missing, compilation warnings may have appeared if the constant string variable was not directly referenced in the user code. This issue has been resolved.

[RTI Issue ID CODEGENII-873]

4.31 Dereference endpoint_data after null check

For C, C++, and modern C++, some static analysis tools detected that the **endpoint_data** parameter in some of the functions of the TypePlugin methods was dereferenced after a null check at the beginning of the functions.

Although dereferencing **endpoint_data** was harmless in this context because **endpoint_data** cannot be NULL, this issue was reported as a warning. This problem has now been resolved. The generated TypePlugin functions (for which the static code analysis reported a warning) now consider a NULL **endpoint_data** an error and return RTI_FALSE.

[RTI Issue ID CODEGENII-880]

4.32 Segfault when calling TypeSupport::deserialize_data_from_cdr_buffer on a buffer containing unknown enum values or union discriminators

A call to **TypeSupport::deserialize_data_from_cdr_buffer** may have produced a segfault if the input buffer contained unknown enum values or union discriminators. For example:

```
enum MyEnumSub {
    unknown,
    ENUM_2,
    ENUM_3
};
enum MyEnumPub {
    unknown,
    ENUM_2,
    ENUM_3,
```

```
    ENUM_4
};
@mutable
struct MyTypePub {
    MyEnumPub myEnum;
};
@mutable
struct MyTypeSub {
    MyEnumSub myEnum;
};
```

If your application called **MyTypePubTypeSupport::serialize_data_to_cdr_buffer** on a sample in which myEnum was set to ENUM_4 and deserialized the output buffer using the API **MyTypeSubTypeSupport::deserialize_data_from_cdr_buffer**, the call to this last API may have produced a segfault.

This problem has been fixed: the call to **MyTypeSubTypeSupport::deserialize_data_from_cdr_buffer** will deserialize ENUM_4 and convert it to unknown.

[RTI Issue ID CODEGENII-881]

4.33 Code Generator failed to generate code when @try_construct annotation used in union discriminator

When the **@try_construct** annotation was used in a union discriminator, *Code Generator* reported an error such as the following one, and did not generate code:

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 2.5.0, please wait ...
ERROR com.rti.ndds.nddsgen.Main test.idl line 7:30 The annotation '@try_construct' is not
applicable for the context: union discriminator.
ERROR com.rti.ndds.nddsgen.Main Fail:  java.lang.Exception: The file couldn't be parsed and the
rawTree wasn't generated
INFO com.rti.ndds.nddsgen.Main Done (failures)
```

This release of *Code Generator* does not support the @try_construct annotation; however, when used it will be ignored, showing just a warning message. You will be able to generate code when using the @try_construct annotation in a union discriminator.

[RTI Issue ID CODEGENII-882]

4.34 Generated code for IDL with const typedef long long did not compile

The generated code for an IDL with a const typedef of "long long" may not have compiled. The generated code for that constant was missing the language-specific letter to indicate that the numerical value was a long long. For example, for the following constant:

```
const UInteger64_T HELLODDS_SIMPLE_LONG = 901298091238;
```

The generated code was:

```
public static final long VALUE = 901298091238;
```

This problem has been resolved. Now the generated code for that example is:

```
public static final long VALUE = 901298091238L;
```

[RTI Issue ID CODEGENII-901]

4.35 Generated code for a constant value with a big integer literal might not have compiled

The generated code for a constant value with a big integer literal might not have compiled in Java or C++ because it was missing the language-required suffix for big literals.

For example, for the following constant:

```
const unsigned long long HELLODDS_SIMPLE_LONG = 901298091238;
```

The generated code did not compile in Java:

```
HELLODDS_SIMPLE_LONG.java:14: error: integer number too large: 901298091238 public static final
long VALUE = 901298091238;
```

This issue has been resolved. Now the corresponding suffix is added when generating code for the literal. In the previous example, the suffix would be as follows:

```
public static final long VALUE = 901298091238L;
```

[RTI Issue ID CODEGENII-932]

4.36 Modified maximum length of sequences and strings when -unboundedSupport is not used, when converting to XML

When *Code Generator* converted an IDL to XML that contained an unbounded sequence, and **-unboundedSupport** was not used, the length of any sequence was -1. Now when **-unboundedSupport** is not used, the length of any unbounded sequence is 100, and the length of any unbounded string is 255. (When **-unboundedSupport** is used, the length of both is still -1.) These values (100 and 255) can be changed by using the options **-sequenceSize** and **-stringSize**.

[RTI Issue ID CODEGENII-936]

4.37 Code generation using -stdString in Traditional C++ was wrong for optional strings

The code generated when using **-stdString** in traditional C++ was wrong for optional strings. For example, when the string was bounded, the generated code did not compile:

```
struct MyStringTypeBounded {
    string<128> m1;
    @optional
```

```
    string<100> m2;
};
```

For optional bounded strings, the generated code did not compile. For optional unbounded strings, the generated code compiled, but the code generated for the copy methods was not correct.

This problem has been resolved.

[RTI Issue ID CODEGENII-942]

4.38 Compiler error when trying to append elements to sample sequence in Ada

When attempting to modify sample sequences in Ada by appending an element to them, compilation failed with an error similar to the following:

```
   [Ada]          dds_collections-example_publisher.adb
dds_collections-example_publisher.adb:120:27: prefix of "Access" attribute must be aliased
gprbuild: *** compilation phase failed
gmake: *** [all] Error 4
```

This was a problem with the generated code for Ada types. This problem has been resolved.

[RTI Issue ID CODEGENII-958]

4.39 using -constructor flag in combination with -optimization set to 1 or 2 may have generated code that didn't compile

Using the **-constructor** flag in combination with **-optimization** <1|2> may have generated code that didn't compile in traditional C++ for IDL containing typedefs.

For example, when generating code for the following IDL:

```
struct MyNestedStruct {
    long m1;
};

typedef MyNestedStruct MyNestedStructTypedef;

struct MyStruct {
    MyNestedStructTypedef m1;
};
```

You may have seen compilation errors like this:

```
MyType.cxx:686:5: error: use of undeclared identifier 'MyNestedStruct_construct_w_params'; did
you mean 'MyNestedStructTypedef_construct_w_params'?
    MyNestedStruct_construct_w_params(&sample->m1,
    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    MyNestedStructTypedef_construct_w_params
MyType.cxx:370:6: note: 'MyNestedStructTypedef_construct_w_params' declared here
void MyNestedStructTypedef_construct_w_params(
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1011]

4.40 C/C++/Modern C++ code generated with optimization level 1 was invalid in some cases

Using the **-optimization** command-line option with value 1 generated invalid code in C, traditional C+, and modern C++ if the IDL had an external typedef. For example:

```
@external
typedef short MyShortExternal;

struct MyTpe {
    MyShortExternal m1;
};
```

The generated code for the previous struct ignored the fact that MyShortExternal should be an external member and mapped the member m1 to **DDS_Short versus DDS_Short*** (in C and C++) or to **dds::-core::external** (in modern C++).

```
typedef struct MyTpe {
    DDS_Short   m1 ;
} MyTpe ;
```

This problem has been fixed.

[RTI Issue ID CODEGENII-1022]

4.41 Incorrect deserialization of extensible types with optional members when receiving a sample with fewer member fields in Java

The deserialization of extensible types with optional members in Java was incorrect when receiving a sample with fewer member fields than the type used in the reading application. The value of the members not present in the sent sample may have been incorrect in the received sample after deserializing. For example, for the following types, the received value for z may have been incorrect.

```
// Publishing type
@appendable
struct example {
    long x;
    @optional
    long y;
};
```

```
// Subscribing type
@appendable
struct example {
    long x;
```

```
    @optional
    long y;
    long z;
};
```

This problem has been resolved.

[RTI Issue ID CODEGENII-1031]

4.42 Incorrect TypeCode name for member fields whose name was a keyword in Java

When generating the TypeCode name for members whose name was a keyword in Java, *Code Generator* added a _ as a prefix to that name. That could cause problems when communicating between a Java application and a C/C++/.Net application using that type.

[RTI Issue ID CODEGENII-1050]

4.43 Code Generator incorrectly generated pub/sub code when all the types were @nested and there was a forward declaration of one of the types

Code Generator incorrectly generated publisher and subscriber code for an IDL that contained all nested types when one of the types was forward declared.

This issue has been fixed. Now *Code Generator* shows an error message like the following that explains that no publisher/subscriber code will be generated for that IDL:

```
INFO com.rti.ndds.nddsgen.Main Running rtiddsgen version 3.0.0, please wait ...
ERROR com.rti.ndds.nddsgen.emitters.CSourceEmitter There isn't any top-level type. Example
files wouldn't be generated
INFO com.rti.ndds.nddsgen.Main Done
```

[RTI Issue ID CODEGENII-1091]

Chapter 5 Known Issues

5.1 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]

5.2 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 {
    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]

5.3 .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]

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

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

[RTI Issue ID BIGPINE-537]

5.5 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]

5.6 -legacyPlugin option not supported on QNX 6.5.1 on PPC when Generating 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 when using the **-legacyPlugin** option 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 code is generated without the **-legacyPlugin** option. (Starting in 3.0.0, the former **-stl** option is the default option. This problem does not occur with the default option.)

[RTI Issue ID CODEGENII-528]

5.7 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{
        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 an underscore character ('_') to the idl module name.

[RTI Issue ID CODEGENII-661]

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

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

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

[RTI Issue ID CODEGENII-800]

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

The default code generator option (formerly **-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 using the **-legacyPlugin** 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 6 Limitations

6.1 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]

6.2 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
                {
                        long 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]

Chapter 7 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/)

Additional information about Third-Party Content contained in the RTI product suite can be found in **RTI_ConnextDDS_3rdPartySoftware_Tools_Services.pdf**.

7.1 Apache Software License Version 2.0

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document. "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the

Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the

appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

7.2 ANTLR 3 License

[The BSD License]

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.