RTI Code Generator 2

for

RTI Connext DDS

Getting Started Guide

Version 2.2.0

Your systems. Working as one.

Printed in U.S.A. First printing.
December 2013.

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

Contents

Welcome to RTI Code Generator 2!

RTI® Connext™ 5.1.0 includes a separate and new implementation of the code generator; this new version significantly improves the performance of the original implementation and makes easier to customize the generated output.

In *Connext* 5.1.0, the new implementation of the code generator, *rtiddsgen2*, coexists with the original, *rtiddsgen*. To generate code using the new implementation, you will use the scripts *rtiddsgen2* or *rtiddsgen2_server* under **NDDSHOME/scripts**.

To recap, there are two versions of *RTI Code Generator*:

- *Code Generator (rtiddsgen)*—the officially supported version that is described in the *RTI Core Libraries and Utilities User's Manual* (see Section 3.6, *Using rtiddsgen*).
- *Code Generator 2 (rtiddsgen2)*—an Early Access Release (EAR) version being testing with *Connext* 5.1.0. This version runs faster that *rtiddsgen* and is easier to modify.

This document describes how to use *rtiddsgen2*.

To see the limitations of *rtiddsgen2* and how it differs with *rtiddsgen*, see Comparing rtiddsgen and rtiddsgen2 (Section 2).

1 Using rtiddsgen2

The *Code Generator* creates the code needed to define and register a user data type with *Connext*. Using either version of the Code Generator (*rtiddsgen* or *rtiddsgen2*) is optional if:

- You are using dynamic types (see Managing Memory for Built-in Types (Section 2.2.8) in the *RTI Core Libraries and Utilities User's Manual*[1]).
- You are using one of the built-in types (see Built-in Data Types (Section 2.2) in the *RTI Core Libraries and Utilities User's Manual*).

To use *rtiddsgen2* you provide a description of a data type in an IDL or XML file. You can define multiple data types in the same type-definition file.

Table 1.1 on page 2-2 (for C, C++, and C++/CLI and C#) and Table 1.2 on page 2-2 (for Java) show the files that *rtiddsgen2* creates for an example IDL file called **Hello.idl**. (The file extension will depend on the chosen language: **.c** for C, **.cxx** for C++, **.cpp** for C++/CLI, **.cs** for C#.)

1. This document is provided with *RTI Connext*. You can also access it from the RTI Community Portal's Documentation page.

On Windows systems: Before running *rtiddsgen2*, run **VCVARS32.BAT** in the same command prompt that you will use to run *rtiddsgen2*.

NOTE: Before using an *rtiddsgen2*-generated makefile to compile an application, make sure the **${NDDSHOME}** environment variable is set as described in the *RTI Core Libraries and Utilities Getting Started Guide*.

Table 1.1 **Files Created by rtiddsgen2 for C, C++, C++/CLI, C# for Example "Hello.idl"**

Generated Files	Description
Required files for the user data type. The source files should be compiled and linked with the user application. The header files are required to use the data type in source. You should not modify these files unless you intend to customize the generated code supporting your type.	
Hello.[c,cxx, cpp] HelloSupport.[c, cxx, cpp] HelloPlugin.[c,cxx, cpp]	Generated code for the data types. These files contain the implementation for your data types.
Hello.h HelloSupport.h HelloPlugin.h	Header files that contain declarations used in the implementation of your data types.
Optional files generated when you use the **-example <*architecture*>** command-line option. You may modify and use these files as a way to create simple applications that publish or subscribe to the user data type.	
Hello_publisher.[c, cxx, cpp, cs]	Example code for an application that publishes the user data type. This example shows the basic steps to create all of the DDS objects needed to send data. You will need to modify the code to set and change the values being sent in the data structure. Otherwise, just compile and run.
Hello_subscriber.[c, cxx, cpp,cs]	Example code for an application that subscribes to the user data type. This example shows the basic steps to create all of the DDS objects needed to receive data using a "listener" function. No modification of this file is required. It is ready for you to compile and run.
Hello.dsw or Hello.sln, Hello_publisher.dsp or Hello_publisher.vcproj, Hello_subscriber.dsp or Hello_subscriber.vcproj	Microsoft Visual C++ or Visual Studio .NET Project workspace and project files, generated only for "i86Win32" architectures. To compile the generated source code, open the workspace file and build the two projects.
makefile_Hello_<*architecture*>	Makefile for non-Windows-based architectures. An example <*architecture*> is linux2.6gcc4.1.1.

Table 1.2 **Files Created by rtiddsgen2 for Java for Example "Hello.idl"**

Data Type	Generated Files	Description
Since the Java language requires individual files to be created for each class, *rtiddsgen2* will generate a source file for every IDL construct that translates into a class in Java.		
Constants	<*Name*>.java	Class associated with the constant
Enums	<*Name*>.java	Class associated with enum type
Structures/ Unions	<*Name*>.java <*Name*>Seq.java <*Name*>DataReader.java <*Name*>DataWriter.java <*Name*>TypeSupport.java	Structure/Union class Sequence class DDS *DataReader* and *DataWriter* classes Support (serialize, deserialize, etc.) class

Table 1.2 **Files Created by rtiddsgen2 for Java for Example "Hello.idl"**

Data Type	Generated Files	Description
Typedef of sequences or arrays	<*Name*>.java <*Name*>Seq.java <*Name*>TypeSupport.java	Wrapper class Sequence class Support (serialize, deserialize, etc.) class
Optional files generated when you use the "-example <*architecture*>" command-line option. You may modify and use these files as a way to create simple applications that publish or subscribe to the user data type.		
Structures/ Unions	<*Name*>Publisher.java <*Name*>Subscriber.java	Example code for applications that publish or subscribe to the user data type. You should modify the code in the publisher application to set and change the value of the published data. Otherwise, both files should be ready to compile and run.
	makefile_Hello_<*architecture*>	Makefile for non-Windows-based architectures. An example <*architecture*> is linux2.6gcc4.1.1.
Structures/ Unions/ Typedefs/ Enums	<*Name*>TypeCode.java (Note: this is *not* generated if you use -notypecode)	Type code class associated with the IDL type given by <*Name*>.

1.1 rtiddsgen2 Command-Line Arguments

There are several command-line options you can pass to *rtiddsgen2*:

```
rtiddsgen2[-autoGenFiles <architecture>]
    [-create <typerfiles|examplefiles|makefiles>]
    [-convertToIDL] | [-convertToXML]
    [-D <name>[=<value>]]
    [-d <outdir>]
    [-enableEscapeChar]
    [-example <architecture>]
    [-help]
    [-I <directory>]
    [[-inputIdl] <IDLInputFile.idl>]|[[-inputXml] <XMLInputFile.xml>]
    [-language <C|C++|Java|Ada|C++/CLI|C#>]
    [-namespace]
    [-noCopyable]
    [-notypecode]
    [-obfuscate]
    [-package <packagePrefix>]
    [-platform <architecture>]
    [-ppDisable]
    [-ppPath <path to the preprocessor>]
    [-ppOption <option>]
    [-replace]
    [-sequenceSize <Unbounded sequences size>]
    [-sharedLib]
    [-stringSize <Unbounded strings size>]
    [-U <name>]
    [-update <typefilese|examplefiles|makefiles>]
    [-use42eAlignment]
    [-V <name>[=<value>]]
    [-verbosity <1-3>]
    [-version]
```

Table 1.1 describes the options (in alphabetical order).

Table 1.1 **Options for rtiddsgen2**

Option	Description
-autoGenFiles <*architecture*>	Updates the autogenerated files, i.e, the typefiles and makefile/project files. Valid options for <*architecture*> are listed in the *RTI Core Libraries and Utilities Platform Notes*. This is a shortcut for: **-update <typefiles> -update<makefiles> -platform <arch>**
-create <typefiles | examplefiles | makefiles>	Creates the files indicated if they do not exist. If the files already exist, the files are not modified and a warning is printed. There can be multiple **-create** options. If both **-create** and **-update** are specified for the same file type, only the **-update** will be applied.
-convertToIDL	Converts the input type description file into IDL format. This option creates a new file with the same name as the input file and a **.idl** extension.
-convertToXML	Converts the input type description file into XML format. This option creates a new file with the same name as the input file and a **.xml** extension.
-D <name>[=<*value*>]	Defines preprocessor macros. Note: On Windows systems, enclose the argument in quotation marks: -D "<*name*>[=<*value*>]"
-d	Generates the output in the specified directory. By default, *rtiddsgen2* will generate files in the directory where the input type-definition file is found.
-enableEscapeChar	Enables use of the escape character '_' in IDL identifiers.
-example <*architecture*>	Generates type files, example files and a makefile. This is a shortcut for: **-create <typefiles> -create<examplefiles> -create<makefiles> -platform <*architecture*>** Valid options for <arch> are listed in the *RTI Core Libraries and Utilities Platform Notes*.
-I <*directory*>	Adds to the list of directories to be searched for type-definition files (IDL or XML files). Note: A type-definition file in one format cannot include a file in another format.
-inputIdl	Indicates that the input file is an IDL file, regardless of the file extension.
IDLInputFile.idl	File containing IDL descriptions of your data types. If -inputIdl is not used, the file must have a '.idl' extension.
-inputXml	Indicates that the input file is a XML file, regardless of the file extension.
-help	Prints out the command line options for *rtiddsgen2*.
-language	Specifies the language to use for the generated files. The default language is C++; you can also choose C, C++/CLI, C#, Java, or Ada.
-namespace	Specifies the use of C++ namespace. (For C++ only. For C++/CLI and C#, it is implied-namespaces are always used.)
-noCopyable	Forces *rtiddsgen2* to put 'copy' logic into the corresponding TypeSupport class rather than the type itself. This option is only used for Java code generation. This option is not compatible with the use of **ndds_standalone_type.jar** (see Section).

Table 1.1 **Options for rtiddsgen2**

Option	Description
-notypecode	Disables type-code support. By using this option, you can generate code that can be used in a standalone manner (see Section). Note: If you are using a large data type (more than 64 K) *and* type code support, you will see a warning when type code information is sent. *Connext* has a type code size limit of 64K. To avoid the warning when working with data types with type codes larger than 64K, turn off type code support by using **-notypecode**.
-obfuscate	Obfuscates the input file to an IDL obfuscated file. Note that if the input type is XML this option will generate an obfuscated IDL.
-package	Specifies the root package into which generated classes will be placed. It applies to Java only. If the type-definition file contains module declarations, those modules will be considered subpackages of the package specified here.
-platform <*architecture*>	Required if -create <typefiles | examplefiles | makefiles> or -update <typefiles | examplefiles | makefiles> is used. Valid options for <*architecture*> are listed in the *RTI Core Libraries and Utilities Platform Notes*.
-ppDisable	Disables the preprocessor.
-ppOption <*option*>	Specifies a preprocessor option. This parameter can be used multiple times to provide the command-line options for the specified preprocessor. See -ppPath <preprocessor executable>.

-ppPath <*preprocessor executable*>	Specifies the preprocessor. If you only specify the name of an executable (not a complete path to that executable), the executable must be found in your **Path**. The default value is "**cpp**" for non-Windows architectures and "**cl.exe**" for Windows architectures.
	If you use **-ppPath** to provide the full path and filename for **cl.exe** or the **cpp** preprocessor, you must also use -ppOption <option>to set the following preprocessor options:
	If you use a non-default path for **cl.exe**, you also need to set:
	`-ppOption /nologo -ppOption /C -ppOption /E -ppOption /X`
	If you use a non-default path for cpp, you also need to set:
	`-ppOption -C`

-replace	Deprecated option. Use -update <typefiles | examplefiles | makefiles> for the proper files (typefiles, examplefiles, makefiles). This option is maintained for backwards compatibility. It allows *rtiddsgen2* to overwrite any existing generated files. If it is not present and existing files are found, *rtiddsgen2* will print a warning but will not overwrite them.
-sequenceSize	Sets the size assigned to unbounded sequences. The default value is 100 elements.
-sharedLib	Generates makefiles that compile with the *Connext* shared libraries (by default the makefile will link with the static libraries)
-stringSize	Sets the size assigned to unbounded strings, not counting a terminating NULL character. The default value is 255 bytes.
-U <*name*>	Cancels any previous definition of <name>.
-update <typefiles | examplefiles | makefiles>	Creates the files indicated if they do not exist. Is the files already exist, this overwrites the file without printing a warning. There can be multiple **-update** options. If both **-create** and **-update** are specified for the same file type, only the **-update** will be applied.

Table 1.1 **Options for rtiddsgen2**

Option	Description
-use42eAlignment	Makes the generated code compatible with *RTI Data Distribution Service* 4.2e. This option should be used when compatibility with 4.2e is required and the topic data types contain double, long long, unsigned long long, or long double members.
-V <*name*< [=<*value*>]	Defines a user variable that can be used in the templates as **$userVarList.name** or **$userVarList.name.equals(value)**. The variables defined with this option are case sensitive.
-verbosity [1-3]	*rtiddsgen2* verbosity: 1: exceptions 2: exceptions and warnings 3: exceptions, warnings and information (Default)
-version	Displays the version of *rtiddsgen2* being used, such as 2.x.y, as well as the version of the templates used, such as ABCD-EFGH-IJKL-NMOP
XMLInputFile.xml	File containing XML descriptions of your data types. If -inputXml is not used, the file must have an **.xml** extension.

1.2 Using Generated Types without RTI Connext DDS (Standalone)

You can use the generated type-specific source and header files without linking the *Connext* libraries or even including the *Connext* header files. That is, the generated files for your data types can be used standalone.

The directory **<NDDSHOME>/resource/rtiddsgen2/standalone** contains the required helper files:

- **include**: header and templates files for C and C++.
- **src**: source files for C and C++.
- **class**: Java jar file.

Note: You must use *rtiddsgen2's* **-notypecode** option to generate code for standalone use.

1.2.1 Using Standalone Types in C

The generated files that can be used standalone are:

- **<idl file name>.c**: Types source file
- **<idl file name>.h**: Types header file

The type plug-in code (**<idl file>Plugin.[c,h]**) and type-support code (**<idl file>Support.[c,h]**) cannot be used standalone.

To use the rtiddsgen2-generated types in a standalone manner:

1. Make sure you use *rtiddsgen2's* **-notypecode** option to generate the code.
2. Include the directory **<NDDSHOME>/resource/rtiddsgen2/standalone/include** in the list of directories to be searched for header files.
3. Add the source files, **ndds_standalone_type.c** and **<idl file name>.c**, to your project.
4. Include the file **<idl file name>.h** in the source files that will use the generated types in a standalone manner.

5. Compile the project using the following two preprocessor definitions:
   a. **NDDS_STANDALONE_TYPE**
   b. The definition for your platform (**RTI_WIN32** or **RTI_UNIX**)

1.2.2 Using Standalone Types in C++

The generated files that can be used standalone are:

- **<idl file name>.cxx**: Types source file
- **<idl file name>.h**: Types header file

The type-plugin code (**<idl file>Plugin.[cxx,h]**) and type-support code (**<idl file>Support.[cxx,h]**) cannot be used standalone.

To use the generated types in a standalone manner:

1. Make sure you use *rtiddsgen2's* **-notypecode** option to generate the code.
2. Include the directory **<NDDSHOME>/resource/rtiddsgen2/standalone/include** in the list of directories to be searched for header files.
3. Add the source files, **ndds_standalone_type.cxx** and **<idl file name>.cxx,** to your project.
4. Include the file **<idl file name>.h** in the source files that will use the *rtiddsgen2* types in a standalone manner.
5. Compile the project using the following two preprocessor definitions:
   a. **NDDS_STANDALONE_TYPE**
   b. The definition for your platform (such as **RTI_WIN32** or **RTI_UNIX**)

1.2.3 Standalone Types in Java

The generated files that can be used standalone are:

- **<idl type>.java**
- **<idl type>Seq.java**

The type code (**<idl file>TypeCode.java**), type-support code (**<idl type>TypeSupport.java**), *DataReader* code (**<idl file>DataReader.java**) and *DataWriter* code (**<idl file>DataWriter.java**) cannot be used standalone.

To use the generated types in a standalone manner:

1. Make sure you use *rtiddsgen2's* **-notypecode** option to generate the code.
2. Include the file **ndds_standalone_type.jar** in the classpath of your project.
3. Compile the project using the standalone types files (**<idl type>.java** and **<idl type>Seq.java**).

2 Comparing rtiddsgen and rtiddsgen2

rtiddsgen2 is an early access release provided with *RTI Connext* 5.1.0. The internal mechanism for generating code has been changed from XSLT templates to Apache's Velocity (http://velocity.apache.org/). This change results in increased performance, and easier-to-understand and updated template files.

The differences with respect to *rtiddsgen* are summarized in the following tables.

Table 2.1 **Languages**

Supported	Unsupported
Ada	*None*
C	
C++	
C++/CLI	
C#	
Java	

Table 2.2 **Command-Line Options**

Supported	Unsupported
-convertToIdl	-convertToCcl
-convertToXml	-convertToCcs
-D <name>[=<value>]	-convertToWsdl
-I <directory>	-convertToXsd
-U <name>	-corba [header file] [-orb <CORBA ORB>]
-d <outdir>	-dataReaderSuffix <Suffix>
-enableEscapeChar	-dataWriterSuffix <Suffix>
-example <arch>	-debug
-help	-expandCharSeq
-inputIdl <IDL file>	-expandOctetSeq
-inputXml <XML file>	-inputWsdl <WSDL file>
-language	-inputXsd <XSD file>
-namespace	-metp
-noCopyable	-optimization <level of optimization>
-notypecode	-typeSequenceSuffix <Suffix>
-package <packagePrefix>	
-ppDisable	
-ppOption <option>	
-ppPath <path to preprocessor>	
-replace	
-sequenceSize <unbounded sequences size>	
-sharedLib	
-stringSize <unbounded strings size>	

Table 2.2 **Command-Line Options**

Supported	Unsupported
-verbosity [1-3]	
-version	
-use42eAlignment	
Additional Options for rtiddsgen2[a]:	
-autoGenFiles <arch>	
-create <typefiles | examplefiles | makefiles>	
-obfuscate	
-platform <arch>	
-sharedLib	
-update <typefiles | examplefiles | makefiles>	
-V <name>[=<value>]	

a. For information on these options, see Table 1.1

Table 2.3 **IDL Types**

Supported	Unsupported
The same IDL types as supported by *rtiddsgen*, except as noted in this table. See Chapter 3 in the *RTI Core Libraries and Utilities User's Manual*.	Bitfields

Table 2.4 **Custom Directives**

Supported	Unsupported
//@key	*None*
//@top-level [true | false]	
//@copy	
//@copy-c	
//@copy-c-declaration	
//@copy-cppcli	
//@copy-cppcli-declaration	
//@copy-declaration	
//@copy-java	
//@copy-java-declaration	
//@copy-java-declaration	
//@Extensibility	
//@ID	
//@resolve-name [true | false]	
Additional Directives for rtiddsgen2:	
//@Optional *(Supported for C, C++ and Java)*	

3 Customizing the Generated Code

rtiddsgen2 allows you to customize the generated code for different languages by changing the provided templates. This version does not allow you to create new output files.

You can load new templates using the command **#parse("<*pathToTemplate*>/template.vm")** in an existing template, where <*pathToTemplate*> is relative to the **<NDDSHOME>/resource/rtiddsgen/templates** folder. If that **template.vm** file contains macros, you can use it within the original template. If **template.vm** contains just plain text without macros, that text will be included directly in the original file.

You can customize the behavior of a template by using the predefined set of variables provided with *rtiddsgen2*. For more information, see the tables in the file **RTI_rtiddsgen_template_variables.xlsx**.

You can add new variables to the templates using the **V <*name*>[=<*value*>]** command-line option when starting *rtiddsgen2*. Refer to a variable in the template as **$userVarList.name** or **$userVarList.name.equals(value)**.

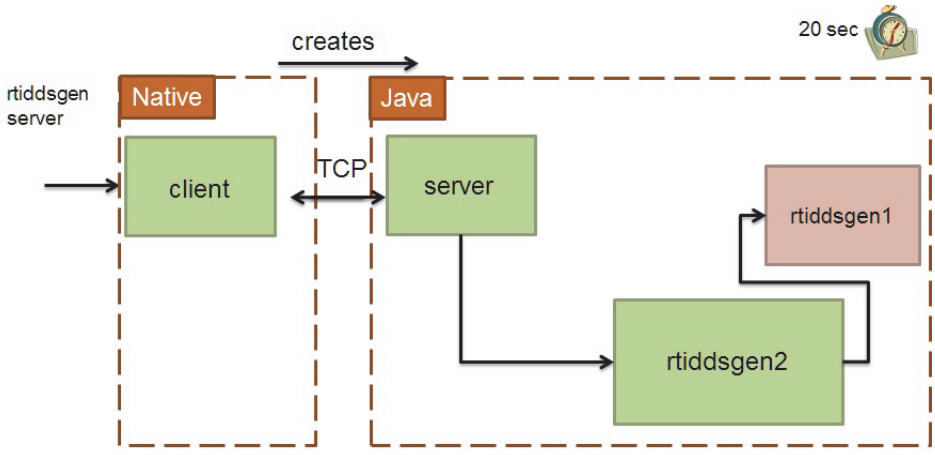
For more information about *rtiddsgen2*'s architecture, see the slides in **RTI_rtiddsgen_architecture.pptx**.

For more information on velocity templates, see http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html.

4 Boosting Performance with Server Mode

Although *rtiddsgen2* has much better performance than *rtiddsgen*, the architecture of the compiler is still Java-based. If you need to invoke the code generator multiple times with different parameters and/or type files, there will be a performance penalty derived from loading the JVM and compiling the velocity templates.

To help with the above scenario, you can run *rtiddsgen2* in server mode. Server mode runs a native executable that opens a TCP connection to a server instance of the code generator that is spawned the first time the executable is run, as depicted below:



When the code generator is used in server mode, JVM is loaded a single time when the server is started; the velocity templates are also compiled a single time.

To invoke the code generator in server mode, use the script **rtiddsgen2_server(.bat)**, which is in the **scripts** directory.

On Windows Systems: To use the **rtiddsgen2_server** script, you must have the Visual Studio 2005 Service Pack 1 redistributable libraries. You can obtain this package from Microsoft or RTI (see the *RTI Core Libraries and Utilities Release Notes* for details).

The *rtiddsgen2* server will automatically stop if it is not used for a certain amount of time. The default value is 20 seconds; you can change this by editing the **rtiddsgen2_server** script and adjusting the value of the parameter, **-n_servertimeout**.