RTI CORBA Compatibility Kit

CORBA-DDS Example Using C++

Instructions

Version 5.0

Your systems. Working as one.

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/

CORBA-DDS Example Using C++

This document will guide you through the steps required to create a CORBA-DDS publisher and a CORBA-DDS subscriber based on an example IDL file. Then it will show you how to adapt those steps to create your own application (the provided example).

This example shows you how to create applications that use CORBA and *RTI® Connext* (formerly *RTI Data Distribution Service*) and share a common set of types for both APIs.

The example includes:

- A CORBA server application.
- An *RTI Connext* subscriber application.
- A combined CORBA client-DDS publisher application. This application uses CORBA and DDS to send text messages to the CORBA server and the *Connext* subscriber. Both receiving applications print the messages to the console.

For more information, please see the *RTI CORBA Compatibility Kit Installation Guide* (**<NDDSHOME>/doc/pdf/RTI_CORBA_Compatibility_Kit_InstallationGuide.pdf**).

1 Introduction

The example is found in **/example/CPP/CORBA** in your *Connext* installation directory. The data type for the test is in **Message.idl**. The application source file is **MessageApp.cxx**. The rest of the source files will be generated by *rtiddsgen* (the *Connext* IDL code generator) and *tao_idl* (TAO's IDL code generator).

2 Setting up Your Environment

If you need general instructions for setting environment variables, please see the *RTI Core Libraries and Utilities Getting Started Guide*.

1. Set the **ACE_ROOT**, **TAO_ROOT** and **LD_LIBRARY_PATH** environment variables.

   For example, if you are using a csh or tcsh shell and ACE+TAO is installed in **/opt/ACE_wrappers**.

   On a 32-bit architecture:

   ```
   setenv ACE_ROOT /opt/ACE_wrappers
   setenv TAO_ROOT ${ACE_ROOT}/TAO
   setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${ACE_ROOT}/lib
   ```

On a 64-bit architecture:

```
setenv ACE_ROOT /opt/ACE_wrappers
setenv TAO_ROOT ${ACE_ROOT}/TAO
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${ACE_ROOT}/lib/x86_64
```

2. Make sure the **NDDSHOME** environment variable is set as described in the *RTI Core Libraries and Utilities Getting Started Guide*. For example, if you are using a csh or tcsh shell and *Connext* is installed in **/local/rti/ndds.5.0.x**:

```
setenv NDDSHOME /local/rti/ndds.5.0.x
```

3 Generating the Source Code

To generate the source code, you will use *rtiddsgen* (*Connext*'s IDL code generator) and *tao_idl* (TAO's IDL code generator).

Note for Cross-compiling: When using cross-compilation, you may need to run *tao_idl* and *rtiddsgen* on the host system. Please refer to **ACE+TAO** documentation for details.

1. Both *tao_idl* and *rtiddsgen* use the **cpp** preprocessor. Make sure that the preprocessor is in your Path (for help, see the *RTI Core Libraries and Utilities Getting Started Guide*).
2. **Copy the IDL file (Message.idl) to a new directory.** In that directory, invoke **tao_idl** on the Message.idl file.

   On a 32-bit architecture:

   ```
   $ACE_ROOT/bin/tao_idl -GT Message.idl
   ```

   On a 64-bit architecture:

   ```
   $ACE_ROOT/bin/x86_64/tao_idl -GT Message.idl
   ```

   This will generate the following CORBA support files:

   - **MessageC.cpp, MessageS.cpp, MessageS_T.cpp**
   - **MessageC.inl, MessageS.inl**
   - **MessageC.h, MessageS.h, MessageS_T.h**
3. Invoke *rtiddsgen* on the **.idl** file with **-corba MessageC.h** (which contains the CORBA type definitions):

   ```
   $NDDSHOME/scripts/rtiddsgen -language C++ \
   -corba MessageC.h -orb <ACE_TAOversion>
   -example <architecture> Message.idl
   ```

   Where **<ACE_TAO*version*>** depends on your version of TAO (for example, **ACE_TAO1.6**) and **<*architecture*>** depends on which architecture you are using (see supported platforms in the *Release Notes*), for example, **i86Linux2.6gcc4.1.2**.

   This will generate the following *Connext* support files:

   - **Message.cxx, MessagePlugin.cxx, MessageSupport.cxx**
   - **Message.h, MessagePlugin.h, MessageSupport.h**
   - **Message_subscriber.cxx, Message_publisher.cxx**
   - **Message.mwc, Message_subscriber.mpc, Message_publisher.mpc** (will be used by TAO's Makefile, Project and Workspace Creator (MPC) tool.
   - **USER_QOS_PROFILES_XML**

Note: If you copied the entire contents of the example directory, remove the ***.mpc** files (or specify **-replace** on the *rtiddsgen* command line)—these files refer to architecture-specific *Connext* libraries.

4 Generating the Makefile

TAO's Makefile, Project and Workspace Creator (MPC) can be used to generate the makefile for the example. Note that Perl must be available to use MPC.

Note for Cross-compiling: When using cross-compilation, you may need to run **mwc.pl** on the host system. Please refer to **ACE+TAO** documentation for details.

Invoke MPC in the same directory you created in Step 2 on Page 2:

```
$ACE_ROOT/bin/mwc.pl -type gnuace Message.mwc
```

This will generate three makefiles:

- For the publisher application: **GNUmakefile.Message_publisher**
- For the subscriber application: **GNUmakefile.Message_subscriber**
- **GNUmakefile**, which executes the above two makefiles.

Note: If you want to use the **.mpc** and **.mwc** files generated by *rtiddsgen* for data types other than the ones in **Message.idl**, you might need to modify your **.mpc** files in order to link against additional TAO libraries.

5 Compiling and Running the Publisher and Subscriber Example

5.1 Compiling the Publisher and Subscriber

In the same directory created in Step 2 on Page 2, run the following:

```
gmake -f GNUmakefile
```

This will execute **GNUmakefile.Message_publisher** and **GNUmakefile.Message_subscriber** and create a **Message_publisher** executable and a **Message_subscriber** executable in your working directory. (For x86_64 architectures: The executables will be in a subdirectory named x86_64 in your working directory.)

Note for Cross-compiling: When using cross-compilation, you will need to have the cross-compile environment set up and run **gmake** on the host system to compile. Please refer to **ACE+TAO** documentation for details.

5.2 Running the Example Publisher and Subscriber

You have now successfully created a publisher and a subscriber for the CORBA data types included in the IDL file, **Message.idl**.

Work in the same directory created in Step 2 on Page 2.

First, set up LD_LIBRARY_PATH in two separate terminals (as in Step 1 on Page 1). Then run the application by entering :

```
Message_subscriber <Domain_ID> <Number_of_Samples>
```
(in the 1st window)

```
Message_publisher <Domain_ID> <Number_of_Samples>
```
(in the 2nd window)

Example output from **Message_subscriber**:

```
time:
    year: 0
    month: 0
    day: 0
    hour: 0
    minute: 0
    second: 0
 msg: "xxxxxxxxxxx"
```

Example output from **Message_publisher**:

```
Writing HelloWorld, count 0
Writing HelloWorld, count 1
Writing HelloWorld, count 2
```

Please refer to the *RTI Core Libraries and Utilities Getting Started Guide* for more on how to run *rtiddsgen*-generated Publisher and Subscriber applications.

6 Compiling and Running the CORBA Example Application

6.1 Using the Generated Files to Build the CORBA Example Application

This section describes how to use the code generated by *tao_idl*/*rtiddsgen* from **Message.IDL** to build a custom application:

1. The custom application code has already been provided for you in **/example/CPP/corba/MessageApp.cxx** in your *Connext* installation directory. Copy that file into the same directory created in Step 2 on Page 2.
2. Copy **Message_publisher.mpc** to a new file named **MessageApp.mpc**. Then edit **MessageApp.mpc**:
   a. Replace the executable name with **MessageApp**:

   ```
   //exename = Message_publisher
   exename = MessageApp
   ```

   b. In the Source_Files section, comment out **Message_publisher.cxx** and add **MesssageApp.cxx**, so that it looks like this:

   ```
   Source_Files {
        MessageC.cpp
        MessageS.cpp
        MessagePlugin.cxx
        Message.cxx
        MessageSupport.cxx
        //Message_publisher.cxx
        MessageApp.cxx
   }
   ```

   c. Save **MessageApp.mpc**.

3. Edit **Message.mwc:** remove **Message_publisher.mpc** and **Message_subscriber.mpc** from the workspace section, replacing them with **MessageApp.mpc**:

```
workspace {
        // Message_publisher.mpc
        // Message_subscriber.mpc
        MessageApp.mpc
}
```

4. Generate the makefile/Visual Studio Project File as described in Section 4.
5. Compile your application:

```
gmake -f GNUmakefile.MessageApp
```

Note for Cross-compiling: When using cross-compilation, you will need to have the cross-compile environment set up and run **gmake** on the host system to compile. Please refer to **ACE+TAO** documentation for details.

6.2 Running the CORBA Example Application

You will need 3 separate command shells. Make sure that each one has the TAO libraries in the dynamic library search path:

❑ Add **$ACE_ROOT/lib** to the **LD_LIBRARY_PATH** environment variable

1. In the first command shell, start the CORBA server application:

   ```
   MessageApp -cr Message.ior
   ```

   The CORBA server will store an IOR (Interoperable Object Reference) in **Message.ior**; this file will be passed to the client application in Step 3.
2. In the second command shell, start the *Connext* subscriber application:

   ```
   MessageApp -nr <domainId>
   ```

   The domain ID is required. Only applications using the same domain ID will communicate with each other.
3. In the third command shell, start the combined CORBA-DDS application, which will publish CORBA and DDS messages:

   ```
   MessageApp -s Message.ior <domainId>
   ```

   The domain ID is required. Use the same value that you used for the subscriber.

   Notice that this is the same filename passed to the CORBA server in Step 1.
4. Type some messages in the combined CORBA-DDS application prompt (command shell 3).

   The messages will be sent to the CORBA Server (command shell 1) *and* the DDS subscriber (command shell 2) using the same types programmatically!

   Note: If you enter **quit** in command shell 3, only the third application will exit; the other two will not be affected.