RTI Data Distribution Service

The Real-Time Publish-Subscribe Middleware

Getting Started Guide

Addendum for Embedded Systems

Version 4.5

Printed in U.S.A. First printing.
June 2010.

Trademarks

Copy and Use Restrictions

Technical Support

Real-Time Innovations, Inc.
385 Moffett Park Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: http://www.rti.com/support

Contents

Chapter 1 Addendum for Embedded Platforms

In addition to enterprise-class platforms like Microsoft Windows and Linux, *RTI® Data Distribution Service* supports a wide range of embedded platforms. This document is a companion to the *Getting Started Guide* especially for users of those platforms. It describes how to configure some of the most popular embedded systems for use with *RTI Data Distribution Service* and to get up and running as quickly as possible. The code examples covered in this document can be generated for your platform(s) using the *rtiddsgen* code generator that accompanies *RTI Data Distribution Service*, as described in the *Getting Started Guide* (Generating Code with rtiddsgen (Section 4.3.2.1)).

This document assumes at least minimal knowledge with the platforms it describes and is not a substitute for the documentation from the vendors of those platforms. For further instruction on the *general* operation of your embedded system, please consult the product documentation for your board and operating system.

Chapter 2 Getting Started on Embedded UNIX-like Systems

This chapter provides instructions on building and running *RTI Data Distribution Service* applications on embedded UNIX-like systems, including QNX® and LynxOS® systems. It will guide you through the process of generating, compiling, and running a Hello World application on an embedded UNIX-like system by expanding on Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*. Please read the following alongside that section.

In the following steps:

- ❏ All commands must be executed in a command shell that has all the required environment variables. For details, see Set Up the Environment on Your Development Machine (Section 3.1.1.1) in the *Getting Started Guide*.
- ❏ You need to know the name of your target architecture (look in your **NDDSHOME/lib** directory). Use it in place of <*architecture*> in the example commands. For example, your architecture might be 'i86LynxOS_SE3.0.0gcc3.4.3'.
- ❏ We assume that you have **gmake** installed. If you have **gmake**, you can use the generated makefile to compile. If you do not have **gmake**, use your normal compilation process. (Note: the generated makefile assumes the correct version of the compiler is already in your path and that **NDDSHOME** is set.)

2.1 Building and Running a Hello World Example

This section describes the basic steps for building and running an *rtiddsgen*-generated example on an embedded UNIX-like target.

1. Create a directory to work in. In this example, we use a directory called **myhello**.
2. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```

3. Use the *rtiddsgen* utility to generate sample code and a makefile. Modify, build, and run the generated code as described in Using Types Defined at Compile Time (Section 4.3.2) in the *Getting Started Guide*:

   For C++:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
gmake -f makefile_HelloWorld_<architecture>
./objs/<architecture>/HelloWorld_subscriber
./objs/<architecture>/HelloWorld_publisher
```

   For Java:

```
rtiddsgen -language Java -example <architecture> HelloWorld.idl
gmake -f makefile_HelloWorld_<architecture>
gmake -f makefile_HelloWorld_<architecture> HelloWorldSubscriber
gmake -f makefile_HelloWorld_<architecture> HelloWorldPublisher
```

   For Java users: The generated makefile deduces the path to the java executable based on the **APOGEE_HOME** environment variable[1], which therefore must be set in order to run the example applications.

2.2 Configuring Automatic Discovery

In most cases, multiple applications—whether on the same host or different hosts—will discover each other and begin communicating automatically. However, in some cases you must configure the discovery service manually. For example, on LynxOS systems, multicast is not used for discovery by default; you will need to configure the addresses it will use. For more information about these situations, and how to configure discovery, see Automatic Application Discovery (Section 4.1) in the *Getting Started Guide*.

1. For example: **$(APOGEE_HOME)/lynx/pcc/ive/bin/j9**

Chapter 3 Getting Started on INTEGRITY Systems

This chapter provides simple instructions on configuring a kernel and running *RTI Data Distribution Service* applications on an INTEGRITY system. Please refer to the documentation provided by Green Hills Systems for more information about this operating system.

This process has been tested on INTEGRITY 5.0.7 and assumes that applications are downloaded dynamically.

For more information on using *RTI Data Distribution Service* on an INTEGRITY system, please see the *RTI Data Distribution Service Platform Notes*.

The first section describes how to build the kernel:

❑ Building the Kernel (Section 3.1)

The next section guides you through the steps to build and run an rtiddsgen-generated example application on an INTEGRITY target.

❑ Building and Running a Hello World Example (Section 3.2)

Before you start, make sure that you know how to:

1. Boot/reboot your INTEGRITY target.
2. Get the serial port output of your target (using telnet, minicom or hyperterminal).

3.1 Building the Kernel

Before you start, you should be familiar with running a kernel on your target.

1. Launch MULTI.
2. Select **File, Create new project**.
3. Choose the INTEGRITY Operating System and make sure the path to your INTEGRITY distribution is correct.
4. Choose a processor family and board name.
5. Click **Next**.
6. Choose Language: **C/C++**.
7. Project type: **INTEGRITY Kernel**.
8. Choose a project directory and name.
9. Click **Next**.
10. In Kernel Options, choose at least: **'TCP/IP stack**'. Everything else can be left to default.
11. In the Project Builder, you should see the following file:

    ***<name of your project>*_default.ld** (under src/resource.gpj).

12. Right-click the file and edit it; the parameters of interest are the following:

```
CONSTANTS
{
    __INTEGRITY_DebugBufferSize = 0x10000
    __INTEGRITY_HeapSize = 0x100000
    __INTEGRITY_StackSize = 0x4000
    __INTEGRITY_DownloadSize = 0x400000
    __INTEGRITY_MaxCoreSize = 0x200000
}
```

    Note that most *RTI Data Distribution Service* applications will require the StackSize and HeapSize parameters to be increased from their default value. The values shown above are adequate to run the examples presented in this document.

13. Once you have changed the desired values, right-click the top-level project and select **Build**.
14. Run the new kernel on your target.

3.2 Building and Running a Hello World Example

This section describes the basic steps for building and running an *rtiddsgen*-generated example on an INTEGRITY target:

- ❏ Generate Example Code and Project File with rtiddsgen (Section 3.2.1)
- ❏ Build the Publish and Subscribe Applications (Section 3.2.2)
- ❏ Connect to the INTEGRITY Target from MULTI (Section 3.2.3)
- ❏ Load the Application on the Target (Section 3.2.4)
- ❏ Run the Application and View the Output (Section 3.2.5)

3.2.1 Generate Example Code and Project File with rtiddsgen

To create the example applications:

1. Create a directory to work in. In this example, we use a directory called **myhello**.
2. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```

3. Use the *rtiddsgen* utility to generate sample code and a project file as described in Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*. Choose either C or C++.

   For C:

```
rtiddsgen -language C -example <architecture> HelloWorld.idl
```

   For C++:

```
rtiddsgen -language ++C -example <architecture> HelloWorld.idl
```

   In your **myhello** directory, you will see that *rtiddsgen* has created a number of source code files (described in Section 3.7 of the User's Manual), additional support files (not listed here), and a project file: **HelloWorld_default.gpj**.
4. Edit the example code to modify the data as described in Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*.

3.2.2 Build the Publish and Subscribe Applications

1. Launch MULTI.
2. Select **File, Open Project Builder** and choose the top-level project file generated by *rtiddsgen*, **HelloWorld_default.gpj**.
3. Right-click the top-level project file in the Project builder and edit it to add the following line:

   ```
   -os_dir=<the path to your INTEGRITY distribution>
   ```

   Note: For Interpeak architectures: Also add the location of your Interpeak libraries:

   ```
   -L<interpeak install dir>/libs
   ```
4. Save your changes.
5. Right-click the top-level project and Build the project. MULTI might ask you if you want to load the changes to your project, you should do so.

3.2.3 Connect to the INTEGRITY Target from MULTI

1. From the MULTI Launcher, click the Connection button and open the Connect option. Your mode should be Download (Download and debug application).
2. Create a custom connection with the following line:

   ```
   rtserv -port udp@<ip address of your INTEGRITY target>
   ```

   (You might be able to see the IP address of your target on the output of its boot sequence.)

   You only have to create your connection once, MULTI will remember it.
3. Make sure your target has booted; *then* select **Connect**. You should see a new window with the Kernel Tasks running on your target.

3.2.4 Load the Application on the Target

1. In the task window, select **Target, Load module**.
2. Browse for your executables; there should be 3 of them in your project directory:
   - **HelloWorld_publisherdd**
   - **HelloWorld_subscriberdd**
   - **posix_shm_manager**

3. Load the **posix_shm_manager** first, it will appear in the **Tasks** window as a separate address space and start running by itself once loaded. It will allow you to use the shared memory transport on your target.

   Note: The default *rtiddsgen*-generated code tries to use shared memory, so unless you have manually disabled it, your application will crash if you do not load the shared memory manager before running the application.

4. Load the publisher, subscriber, or both. They should appear in separate address spaces in the Tasks window.

3.2.5 Run the Application and View the Output

1. Select the task called "Initial" in your application's address space in the Tasks window; you can either click the play button to run it, or click the debug button to debug it.

   Note that with some versions of INTEGRITY, it is difficult to pass arguments to applications. Arguments can always be hard-coded in your application before compiling it. To quickly experiment with multiple runs of the application with different arguments, one option is to run your application within the debugger. Then you can set a breakpoint before the arguments are used and change them at that point.

2. From the Tasks window, select **Target, Show Target Windows**. This will show you the standard output of your target.

   Some errors messages may still go through the serial port, so you should leave your serial port connection open and monitor it as well.

To reboot the target:

Go to your serial port connection monitor and type '**rset**'.

Chapter 4 Getting Started on VxWorks 6.x Systems

This chapter provides simple instructions to configure a kernel and run *RTI Data Distribution Service* applications on VxWorks 6.x systems. Please refer to the documentation provided by Wind River Systems for more information on this operating system. See also the *RTI Data Distribution Service Platform Notes*.

This chapter will guide you through the process of generating, compiling, and running a Hello World application on vxWorks 6.x systems by expanding on Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*; please read the following alongside that section.

The first section describes how to build the kernel:

- ❏ Building the Kernel (Section 4.1)

The next section guides you through the steps to generate, modify, build, and run the provided example HelloWorld application on a VxWorks target:

- ❏ Building and Running a Hello World Example (Section 4.2)

4.1 Building the Kernel

Before you start, you should be familiar with running a kernel on your target.

1. Launch Workbench.
2. Select **File, New, VxWorks Image Project**
3. Give your project a name.
4. Choose the board support package according to your hardware.

5. For the Tool chain, select **GNU**.
6. For the Profile, select **PROFILE_DEVELOPMENT**.

   Leave everything else at its default setting.
7. For VxWorks 6.4 and below: Once the project is created, double-click **Kernel Configuration** and add at least the following modules:
   - **ZBUF Socket** (under **Network Components, Network Socket Components**)

     (The *RTI Data Distribution Service* libraries for VxWorks Kernel Mode use ZBUF sockets. If you do *not* add this module to the kernel, you will see undefined symbols when loading the *RTI Data Distribution Service* application on the target.)
   - **IGMP v4** (under **Network Components, Network Protocol Components, Network IPv4 Components**)

     This will enable multicast for the target.

   To add a module to the kernel, right-click **include**.

   Included modules have a dark blue box, modules not included have light blue box.
8. Compile the Kernel by right-clicking the project and selecting **Build project**.

   The Kernel and associated symbol file will be found under **<your project directory>/default/**.

4.2 Building and Running a Hello World Example

This section will guide you through the steps required to successfully run an *rtiddsgen*-generated example application on a VxWorks 6.x target, using kernel mode or RTP mode:

- ❏ **Kernel Mode**: see Section 4.2.2
- ❏ **RTP Mode**: see Section 4.2.3

4.2.1 Generate Example Code and Makefile with rtiddsgen

To create the example applications:

1. Create a directory to work in. In this example, we use a directory called **myhello**.
2. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```

3. Use the *rtiddsgen* utility to generate sample code and a makefile as described in Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*.Choose either C or C++.

   Note: The architecture names for Kernel Mode and RTP Mode are different.

   For C:

```
rtiddsgen -language C -example <architecture> HelloWorld.idl
```

   For C++:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
```

   For Java:

```
rtiddsgen -language Java -example <architecture> HelloWorld.idl
```

   Edit the generated example code as described in Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*.

4.2.2 Building and Running an RTI Data Distribution Service Application as a Kernel Task

There are two ways to build and run your *RTI Data Distribution Service* application:

- ❏ Using Command Line (Section 4.2.2.1)
- ❏ Using Workbench (Section 4.2.2.2)

4.2.2.1 Using Command Line

1. Set up your environment with the **wrenv.sh** script in the VxWorks base directory.
2. Build the Publisher and Subscriber modules using the generated makefile. This makefile will work out of the box on a Solaris host; you will have to modify the compiler and linker paths for Linux and Windows hosts. To use dynamic linking, remove the *RTI Data Distribution Service* libraries from the link objects.

(Note: steps 4, 5 and 6 can be replaced by establishing a telnet connection to the VxWorks target. In that case, Workbench does not need to be used and both the Host Shell and Target Console will be redirected to the telnet connection. Once in the C interpreter (prompt '->' in the shell) you can type 'cmd' and then 'help' for more information on how to load and run applications on your target.)

3. Launch Workbench.
4. Make sure your target is running VxWorks.
5. Connect to the target with the target manager and open a host shell (and a Target Console Tool to look at the output, both are found by right-clicking the connected target in the **Target Tools** sub-menu.)
6. In the host shell: if using static linking, load the single **.so** file produced by the build:

```
>cd "directory"
>ld 1 < HelloWorld_subscriber.so
```

   If using dynamic linking: to avoid unresolved symbols warnings, load the libraries first, in this order: **libnddscore.so**, then **libnddsc.so**, then **libnddscpp.so**; *then* load your **.so** file.

7. Run the **subscriber_main** or **publisher_main** function. For example:

```
>taskSpawn "sub", 255, 0x8, 150000, subscriber_main, 38, 10
```

   In this example, 38 is the domain ID and 10 is the number of samples.

Note: If you plan on running more than one *RTI Data Distribution Service* application as a Kernel task (for example, a publisher and a subscriber), use dynamic linking to avoid symbol conflicts.

4.2.2.2 Using Workbench

1. Using Workbench, create a new Downloadable Kernel Module Project, choose the **Managed build** option and the **PPC32gnu** architecture for a PowerPC target, or **PENTIUMgnu** for an i86 target.
2. Copy the source files and headers generated by *rtiddsgen* into the project directory.
3. Right-click the project in Workbench, then select **Refresh** to see the files.
4. Right-click the project and go to **Properties**, then go to **Build Properties**.
5. In the **Build Macros** tab:
   - Add to DEFINES: **-DRTI_VXWORKS**

- Add to CC_ARCH_SPEC: **-mlongcall** (only needed for PowerPC targets)
- If you are use static linking:

  Add to LIBPATH: **-L/{your path to the *RTI Data Distribution Service* libraries, make sure they're the Kernel ones}**

  Add to LIBS: **-lnddscppz -lnddscz -lnddscorez** *(in that order)*

  (If you are using dynamic linking, there are no changes required to LIBPATH or LIBS.)

6. In the **Build Paths** tab, click **Generate**. *Make sure all the Substitute paths are unchecked.*
7. Click **Next**, then manually add the folders containing the *RTI Data Distribution Service* header files using the **Add Folder** option: **$(NDDSHOME)/include** and **$(NDDSHOME)/include/ndds**.
8. Click **Resolve All** and make sure there are no more **Unresolved Include Directives**.
9. Click **Apply** to save the changes, then click **OK** to exit the properties menu.
10. Right-click the project, then select **Build**.
11. Run the application as described starting in Step 3 on page 4-4, except you should load **HelloWorld.out** instead of **HelloWorld_subscriber.so** when you get to Step 6 on page 4-4.

4.2.3 Building and Running an RTI Data Distribution Service Application as a VxWorks Real-Time Process (RTP)

There are two ways to build and run your *RTI Data Distribution Service* RTP application:

- ❏ Using the Command Line (Section 4.2.3.1)
- ❏ Using Workbench (Section 4.2.3.2)

4.2.3.1 Using the Command Line

1. Generate the source files and the makefile with *rtiddsgen*.

   Note: The architecture names for Kernel Mode and RTP Mode are different.

   Please refer to the *RTI Data Distribution Service User's Manual* for more information on how to use *rtiddsgen*.
2. Set up your environment with the **wrenv.sh** script in the VxWorks base directory.

3. Build the Publisher and Subscriber modules using the generated makefile. This makefile will work out of the box on a Solaris host; you will have to modify the compiler and linker paths for Linux and Windows hosts.

   (Note 1: steps 4 to 10 can be replaced by establishing a telnet connection to the VxWorks target. In that case, Workbench does not need to be used and both the Host Shell and Target Console will be redirected to the telnet connection. Once in the C interpreter (prompt '->' in the shell) you can type 'cmd' and then 'help' for more information on how to load and run applications on your target.)

   (Note 2: if you want to dynamically link your RTP to the RTI libraries (VxWorks 6.3 and above only), make the following modifications the generated makefile:

   ```
   LIBS = -L$(NDDSHOME)/lib/<architecture>  -non-static -lnddscpp \
          -lnddsc -lnddscore $(syslibs_<architecture>)
   ```

   Then add to the **LD_LIBRARY_PATH** environment variable the path to your RTI libraries as well as the path to **libc.so.1** of your VxWorks installation to launch your RTP successfully.)

4. Launch Workbench.
5. Make sure your target is running VxWorks.
6. Connect to the target with the target manager and open a host shell and a Target Console Tool to look at the output. Both are found by right-clicking the connected target in the **Target Tools** sub-menu.
7. Right-click your target in the Target Manager window, then select **Run Real Time Process**.
8. Set the **Exec Path on Target** to the **HelloWorld_subscriber.vxe** or the **HelloWorld_publisher.vxe** file created by the build.
9. Set the arguments (domain ID and number of samples, using a space separator).

   A Stack size of 0x100000 should be sufficient. If your application doesn't run, try increasing this value.
10. Click **Run**.

4.2.3.2 Using Workbench

1. Create a new Real Time Process Project; make sure the **Managed build** option is checked and choose the **PPC32gnu** architecture for a PowerPC target, or **PENTIUMgnu** for an i86 target.

2. Copy the source files and headers generated by *rtiddsgen* into the project directory. There can only be one **main()** in your project, so you must choose between a subscriber or a publisher. If you want to run both, you will need two create 2 different projects.
3. Right-click the project in Workbench, then select **Refresh** to see the files.
4. Right-click the project and go to **Properties**, then select **Build Properties**.
5. In the **Build Macros** tab, add:
   - to DEFINES: **-DRTI_VXWORKS**
   - to CC_ARCH_SPEC: **-mlongcall** (only needed on PowerPC targets)
   - to LIBPATH: **-L/{your path to the ndds libraries, make sure they are the RTP ones}**
   - to LIBS:
     - for static linking:
       ```
       -lnddscppz -lnddscz -lnddscorez (in that order)
       ```
     - for dynamic linking (VxWorks 6.3 and above only):
       ```
       -non-static -lnddscpp -lnddsc -lnddscore (in that order)
       ```
6. In the Build Paths tab, click **Generate**. *Make sure all the Substitute paths are unchecked.*
7. Click **Next**, then manually add the folders containing the *RTI Data Distribution Service* header files using the **Add Folder** option: **$(NDDSHOME)/include** and **$(NDDSHOME)/include/ndds**.
8. Click **Resolve All** and make sure there are no more Unresolved Include Directives
9. Click **Apply** to save the changes, then click **OK** to exit the properties menu.
10. Right-click the project, then select **Build**.
11. Run the application as described starting in Step 4 on page 4-6.

Chapter 5 Getting Started on Windows CE Systems

This chapter provides instructions on building and running *RTI Data Distribution Service* applications on a Windows CE system. It will guide you through the process of generating, compiling, and running the Hello World application on a Windows CE system.

In the following steps:

- ❏ All commands must be executed in a command shell that has all the required environment variables. For details, see Set Up the Environment on Your Development Machine (Section 3.1.1.1) in the *Getting Started Guide*.
- ❏ You need to know the name of your target architecture (look in your **%NDDSHOME%\lib** directory). Use it in place of <*architecture*> in the example commands. Your architecture might be 'i86WinCE6.0VS2005' or 'armv4WinCE6.0VS2005'.
- ❏ We assume that you have Visual Studio 2005 installed. If you have Visual Studio, you can use the solution file generated by the *rtiddsgen* utility to compile your application. If you do not have Visual Studio, use your normal process for compiling.

5.1 Building and Running a Hello World Example

To create the example applications:

1. Create a directory to work in. In this example, we use a directory called **myhello**.

2. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

```
struct HelloWorld {
    string<128> msg;
};
```

3. Use *rtiddsgen* to create example code and a solution file as described in Generating Code with rtiddsgen (Section 4.3.2.1) in the *Getting Started Guide*.

   For C, enter:

```
rtiddsgen -language C -example <arch> HelloWorld.idl
```

   For C++, enter:

```
rtiddsgen -language C++ -example <architecture> HelloWorld.idl
```

   You should see the following message, which is normal output from the **cl.exe** command line compiler:

```
rtinddsgenXXX.cc
```

   Edit and build the generated example applications as described in the *Getting Started Guide* section noted above.

4. Start the subscriber application, **HelloWorld_subscriber**. You can run it from Visual Studio, or copy it to the target and then run it from a command prompt on the target.

   The application executables are in **myhello\objs\<*architecture*>**.

   If you copy them to **'\Hard Disk\myhello\objs\<*architecture*>'** on the target, to start the application:

```
cd "Hard Disk\myhello\objs\<architecture>"
HelloWorld_subscriber
```

   In this command window, you should see that the subscriber is waking up every 4 seconds to print a message:

```
HelloWorld subscriber sleeping for 4 sec...
HelloWorld subscriber sleeping for 4 sec...
HelloWorld subscriber sleeping for 4 sec...
```

5. Open another command prompt window and start the publisher application, **HelloWorld_publisher**.

```
cd "Hard Disk\myhello\objs\<architecture>"
HelloWorld_publisher
```

In this second (publishing) command window, you should see:

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

Look back in the first (subscribing) command window. You should see that the subscriber is now receiving messages from the publisher:

```
HelloWorld subscriber sleeping for 4 sec...
msg: "Hello World! {0}"
HelloWorld subscriber sleeping for 4 sec...
msg: "Hello World! {1}"
HelloWorld subscriber sleeping for 4 sec...
msg: "Hello World! {2}"
```