RTI Connext DDS Core Libraries

Getting Started Guide

Addendum for Embedded Systems

Version 6.1.0

Printed in U.S.A. First printing.
April 2021.

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 Addendum for Embedded Platforms

In addition to enterprise-class platforms like Microsoft Windows and Linux, *RTI® Connext® DDS* supports a wide range of embedded platforms. This document is especially for users of those platforms. It describes how to configure some of the most popular embedded systems for use with *Connext DDS* 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 *RTI Code Generator* (*rtiddsgen)*, which accompanies *Connext DDS*.

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 QNX Embedded Systems

This document provides instructions on building and running *Connext DDS* applications on embedded systems such as QNX® systems. It will guide you through the process of generating, compiling, and running a Hello World application on an embedded QNX system by expanding on Hands-On 1 of *Introduction to Publish/Subscribe*, in the RTI Connext DDS 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 Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext DDS 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 'armv7aQNX6.6.0qcc_cpp4.7.3'.
- We assume that you have **make** installed. If you have **make**, you can use the generated makefile to compile. If you do not have **make**, 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 target such as QNX.

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 shown below.

   For C++:

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

   For Java:

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

   Edit the example code to add the line **sprintf(instance->msg, "Hello World! (%d)", count);** as follows:

   ```
   for (count=0; (sample_count == 0) || (count < sample_count); ++count) {

       printf("Writing HelloWorld, count %d\n", count);

       /* Modify the data to be written here */
       sprintf(instance->msg, "Hello World! (%d)", count);

       /* Write data */
       retcode = HelloWorldDataWriter_write(
           HelloWorld_writer, instance, &instance_handle);
       if (retcode != DDS_RETCODE_OK) {
           fprintf(stderr, "write error %d\n", retcode);
       }

       NDDS_Utility_sleep(&send_period);

   }
   ```

4. With the NDDSHOME environment variable set, build the Publisher and Subscriber modules using the generated makefile.

   ```
   make -f makefile_HelloWorld_<architecture>
   ```

   For details on setting up the NDDSHOME environment variable, see Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext DDS Getting Started Guide.

   After compiling, you will find the application executables in **myhello/objs/<*architecture*>**.

5. Connect to the QNX target (using ssh, for example) and start the subscriber application, **HelloWorld_subscriber**.

```
HelloWorld_subscriber
```

In this shell, you should see that the subscriber is waking up every 4 seconds to print a message. Here is a C++ example:

```
No data after 1 second
No data after 1 second
No data after 1 second
```

6. Connect to the QNX target and start the publisher application, **HelloWorld_publisher**.

```
HelloWorld_publisher
```

In this second (publishing) shell, you should see:

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

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

For example, in C++:

```
Received data
   msg: "Hello World! (0)"
Received data
   msg: "Hello World! (1)"
Received data
   msg: "Hello World! (2)"
```

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. See information about setting discovery peers in the "Troubleshooting" section of *Introduction to Publish/Subscribe*, in the RTI Connext DDS Getting Started Guide.

Chapter 3 Getting Started on INTEGRITY Systems

This section provides simple instructions on configuring a kernel and running *Connext DDS* applications on an INTEGRITY system. These instructions assume that the application module will be dynamically downloaded. Please refer to the documentation provided by Green Hills Systems for more information about this operating system.

For more information on using *Connext DDS* on an INTEGRITY system, please see the INTEGRITY section of the RTI Connext DDS Core Libraries Platform Notes.

The first section describes 3.1 Building the Kernel below.

The next section guides you through the steps to build and run an rtiddsgen-generated example application on an INTEGRITY target: 3.2 Building and Running a Hello World Example on the next page.

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 *Connext DDS* 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:

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. 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 *Generated Files*, in the RTI Code Generator User's Manual), additional support files (not listed here), and a project file: **HelloWorld_default.gpj**.
4. For C only, edit the example code (to add the line **sprintf(instance->msg, "Hello World! (%d)", count);**) as follows:

```
for (count=0; (sample_count == 0) || (count < sample_count); ++count) {

    printf("Writing HelloWorld, count %d\n", count);

    /* Modify the data to be written here */
    sprintf(instance->msg, "Hello World! (%d)", count);

    /* Write data */
    retcode = HelloWorldDataWriter_write(
        HelloWorld_writer, instance, &instance_handle);
    if (retcode != DDS_RETCODE_OK) {
        fprintf(stderr, "write error %d\n", retcode);
    }

    NDDS_Utility_sleep(&send_period);

}
```

   You do not need to edit the C++ code, *rtiddsgen* has added **sprintf(instance->msg, "Hello World! (%d)", count);** for C++ already.

3.2.2 Build the Publish and Subscribe Applications

1. In a plain text editor, edit the top-level project file that was generated by *rtiddsgen*, **HelloWorld_default.gpj**, so that it points to the path to your INTEGRITY distribution:
   - For INTEGRITY 5 systems:

     Under **[Project]**, add the argument **-os_dir=**<*path to your INTEGRITY distribution*>
   - For INTEGRITY 10 or 11 systems:

     Set **macro __OS_DIR=**<*path to your INTEGRITY distribution*>
2. Save your changes.
3. Launch MULTI.
4. Open the top-level project file, **HelloWorld_default.gpj**, in MULTI:
   - For INTEGRITY 5 systems:

     Select **File, Open Project Builder**, then open the project file from there.
   - For INTEGRITY 10 or 11 systems:

     Select **Components, Open Project Manager**, then open the project file from there.
5. Right-click on the top-level project and build the project.

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:

   For targets that only support the older INDRT connection mechanism:

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

   For targets that support the newer INDRT2 connection mechanism:

   ```
   rtserv2 -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/7 Systems

This section provides simple instructions to configure a kernel and run *Connext DDS* applications on VxWorks 6.x/7 systems. Please refer to the documentation provided by Wind River Systems for more information on this operating system.

This chapter will guide you through the process of generating, compiling, and running a Hello World application on VxWorks 6.x/7 systems by expanding on the VxWorks section of the RTI Connext DDS Core Libraries Platform Notes; please read the following alongside that section.

The first two sections describe how to build a VxWorks Source Build (VSB) and the kernel:

- 4.1 Building the VSB on the next page (only needed for VxWorks 7 systems)
- 4.2 Building the Kernel on page 13

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

- 4.3 Building and Running a Hello World Example on page 18

For tips on using RTI DDS Ping and Spy, see 4.4 Using DDS Ping and Spy on page 34.

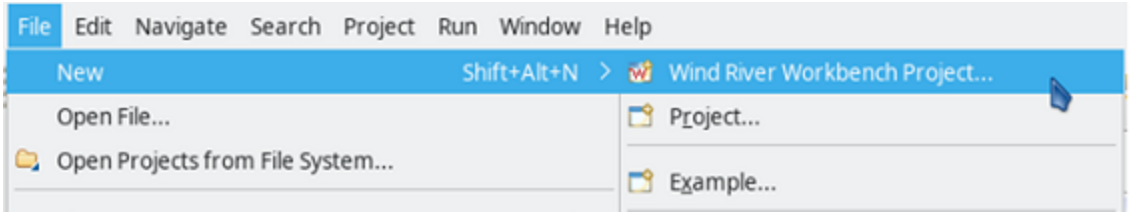
4.1 Building the VSB

This section explains how to build a VxWorks Source Build (VSB), which is required in order to build your own kernels and applications with VxWorks 7. If you are using VxWorks 6.x, you can skip this section.

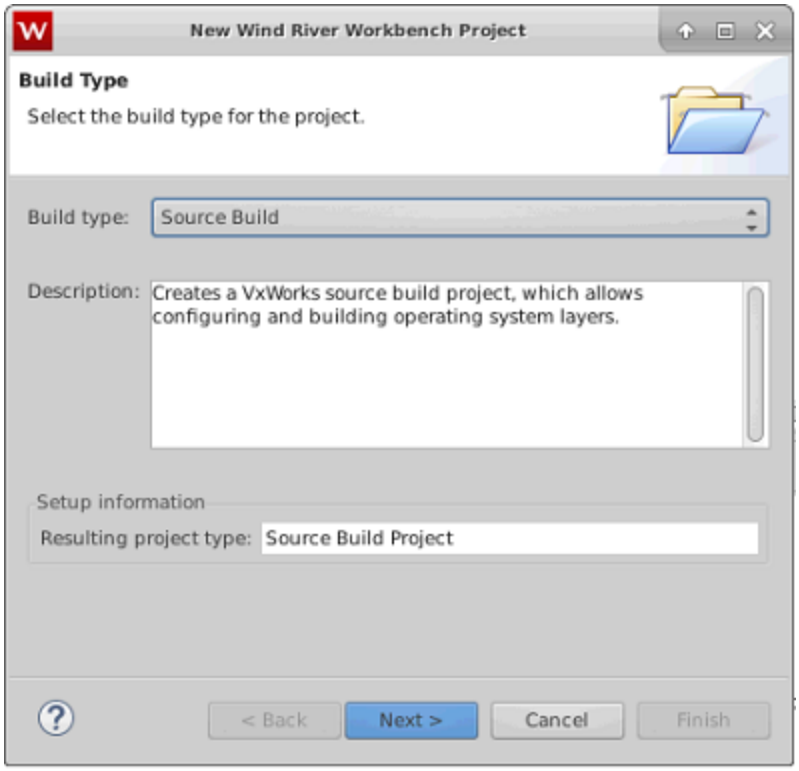
The following steps use the VSB defaults. For further information and special customizations, please refer to Wind River's documentation:
https://docs.windriver.com/bundle/Configuration_and_Build_Guide_Edition_9_1/page/1597954.html

Before you start, you should be familiar with your hardware, as you will need to select a BSP and other hardware-specific settings. This document uses an Intel BSP as an example.

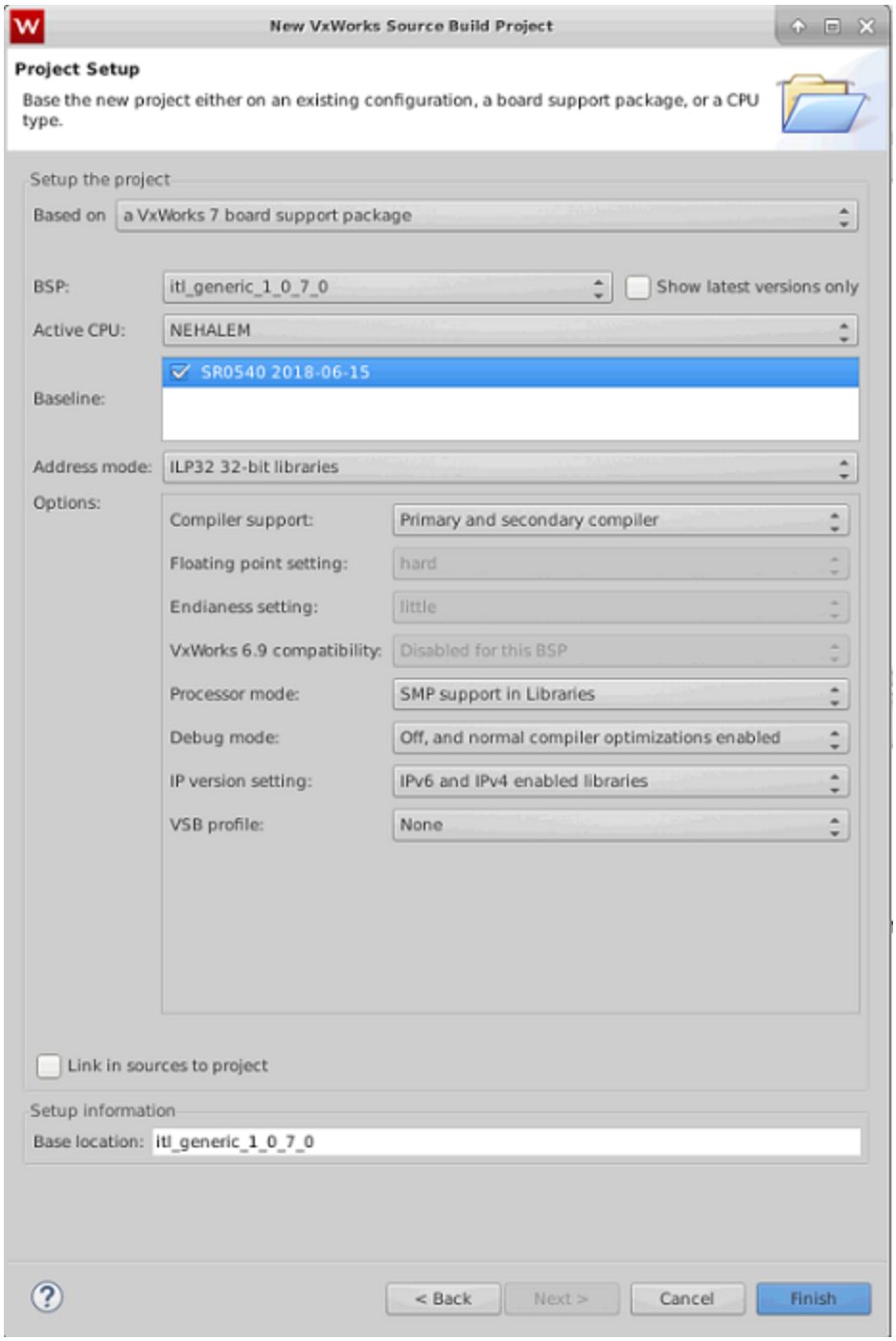
1. Launch Workbench.
2. Select **File, New, Wind River Workbench Project**.



3. For the Build type, select **Source Build**.



4. Set your project name and click **Next**.
5. Configure your VSB. Set your BSP, the CPU, addressing mode, compiler, SMP, etc., according to your platform. When you are done, click **Finish**.



6. After you finish, build the VSB as you would any other project.

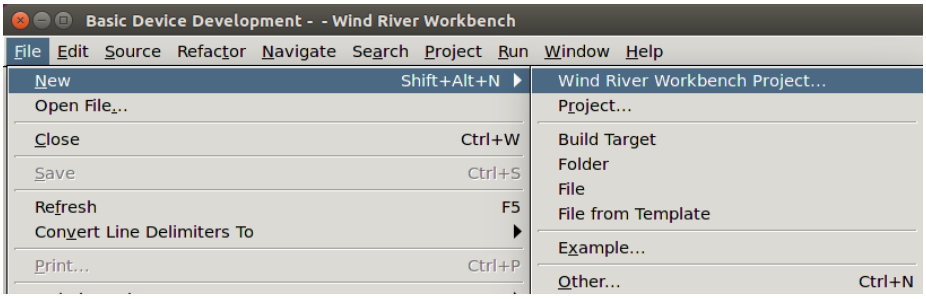
4.2 Building the Kernel

This section explains how to build a kernel capable of loading *Connext DDS* libraries. *Connext DDS* libraries require that certain components are added to the default list in the VxWorks kernel, as outlined in the following steps.

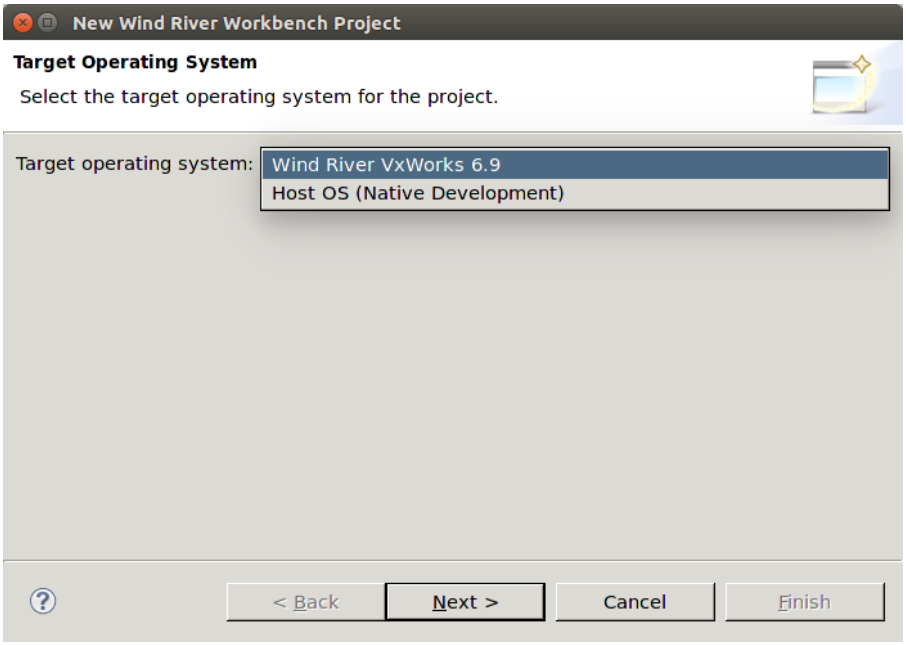
Before you start, you should be familiar with building and deploying a default working kernel on your target.

Note: The following steps might vary slightly depending on your chosen version of VxWorks.

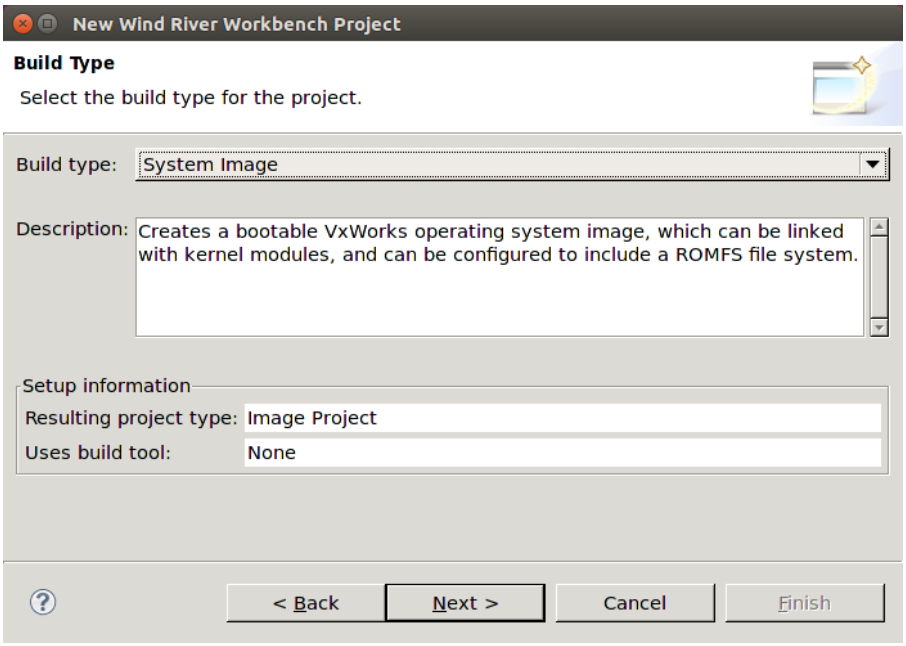
1. Launch Workbench.
2. Select **File, New, Wind River Workbench Project**.



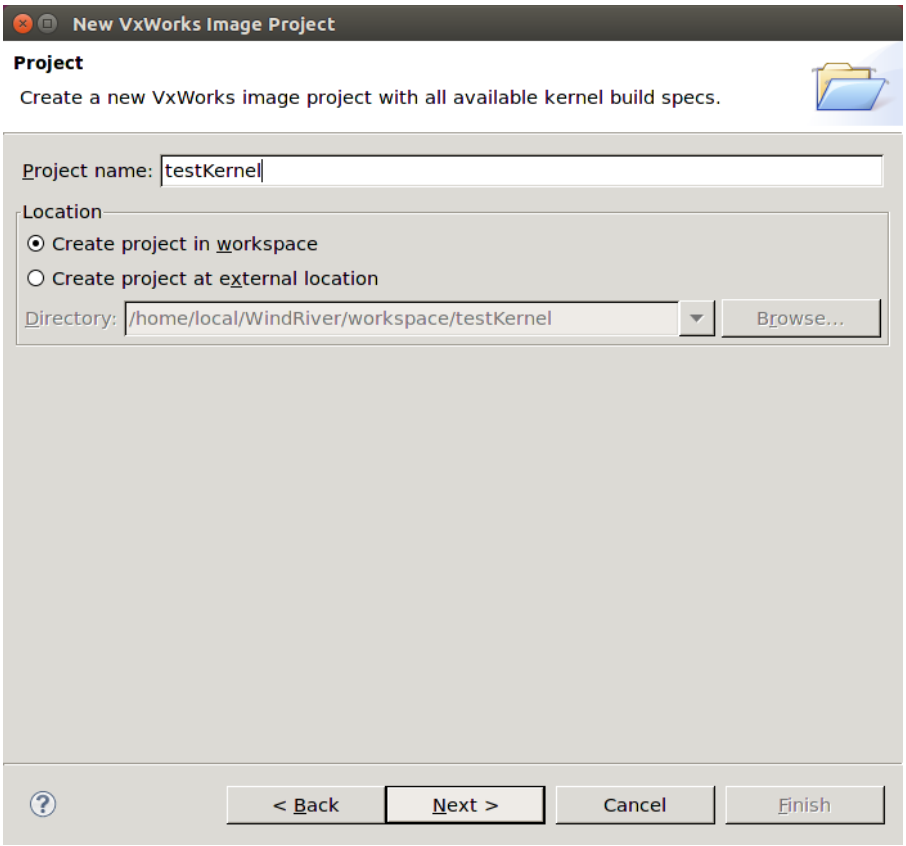
3. Select the desired **Target operating system**; click **Next**.



4. When prompted to choose a **Build type**, select **System Image** (this may be **Kernel Image** or **VxWorks Image** depending on your version of VxWorks); click **Next**.



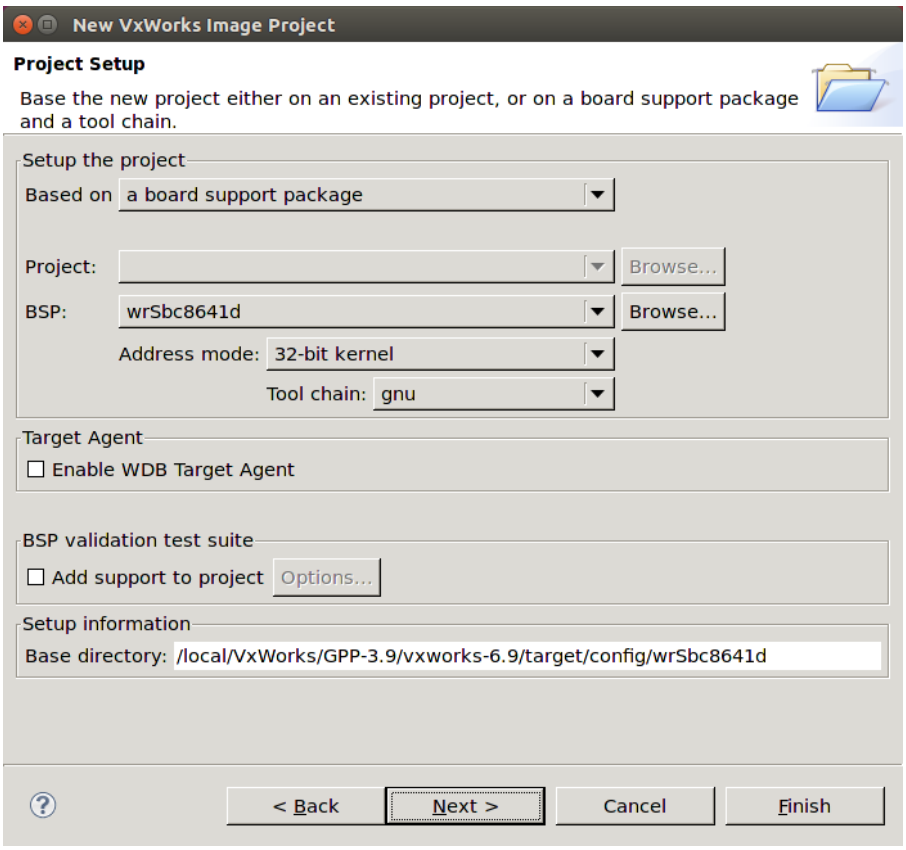
5. Give your project a name; click **Next**.



6. **VxWorks 6.x**: In Project Setup, choose a board support package (BSP) based on your hardware. If available, select the correct Address mode.

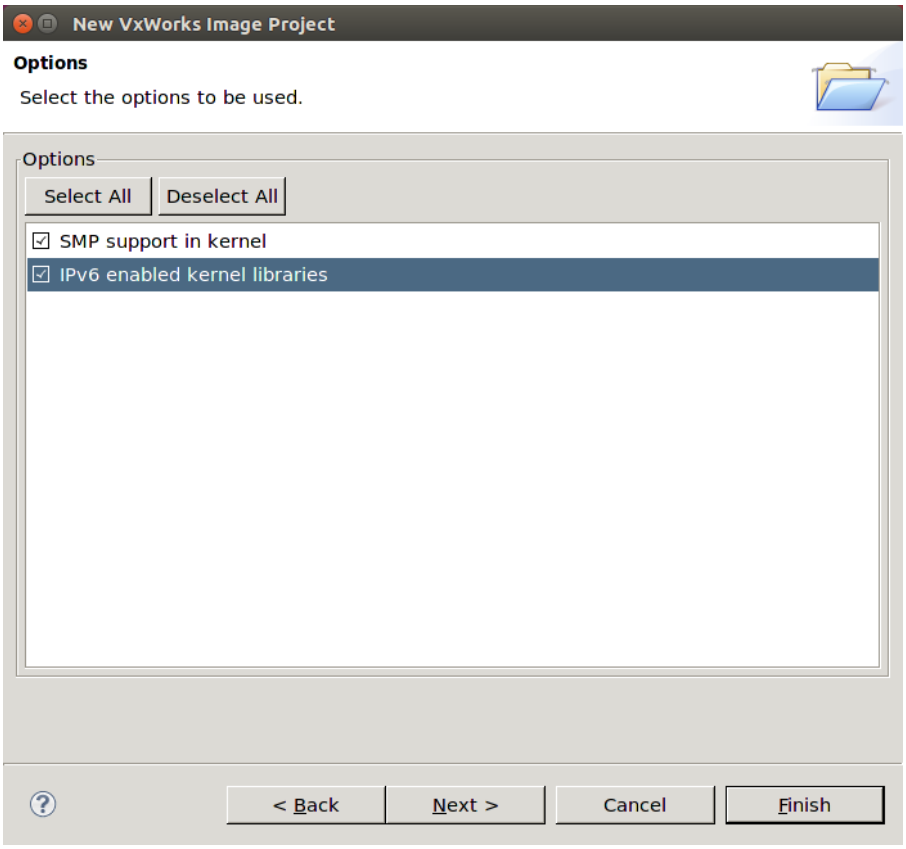
   VxWorks 7: In Project Setup, for the **Based on** field, choose **a source build project**. For the **Project**, choose the VSB you created and built in the previous section. The BSP, SMP support and other options will be correctly populated from the VSB configuration.

   For the **Tool chain** option, select **GNU**; click **Next**.

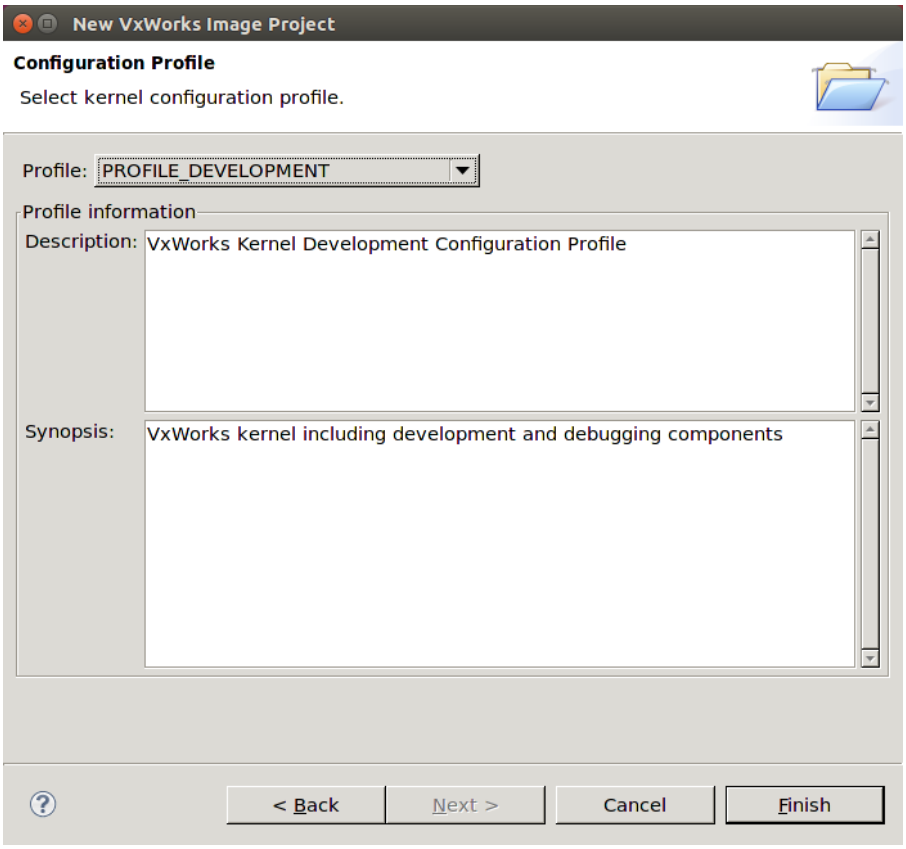


7. In **Options**, select **SMP support in kernel** if your BSP supports it and you want to enable symmetric multi-processing capability in the kernel.

   Select **IPv6 enabled kernel libraries** if your architecture supports IPv6 (See the VxWorks section of the RTI Connext DDS Core Libraries Platform Notes to check if your architecture supports IPv6); click **Next**.



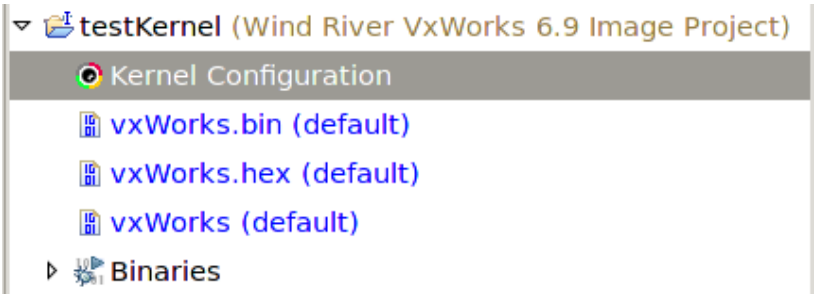
8. Optionally, select a configuration profile from the drop-down menu.



9. Leave everything else at its default setting. Click **Finish**.

   Your project will be created at this time.

10. From the Project Explorer, open **Kernel Configuration**.



11. Add **Operating System Components, Kernel Components, _thread variables support**.
12. Make sure you have the following components enabled: INCLUDE_TIMESTAMP, INCLUDE_SHARED_DATA, INCLUDE_TLS.

    Note: If you are unwilling or unable to build shared-memory support into your kernel, see the VxWorks section of the RTI Connext DDS Core Libraries Platform Notes.

13. If you plan to use any *Connext DDS* C++ API, you will need to include the FOLDER_CPLUS section in your kernel (the underlying kernel components may vary depending on the VxWorks version). This includes Traditional and Modern C++ APIs and Request/Reply C++ APIs.
14. If you want support for RTP shared libraries, you need to add the component **INCLUDE_SHL**. Note that shared libraries are not supported in all VxWorks architectures.
15. If you plan on accessing your target via the network, you may need the following modules:
    - **Telnet Server** (under Network Components, Applications, Telnet Components)

      This will allow you to telnet into the target.
    - **NFS client all** (under Operating System Components, IO System Components, NFS components)

      This will allow you to see networked file systems from the target (contact your system administrator to find out if you have them set up).
16. If you are running applications in RTP mode, you may increase **Operating System components, Real Time Processes components, Number of entries in an RTP fd table** from the default value of 20 to a higher value such as 256. This will enable you to open more sockets from an RTP application.
17. Compile the Kernel by right-clicking the project and selecting **Build Project.**

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

4.3 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/7 target using kernel mode or RTP mode.

4.3.1 Generate Example Code and Makefile with rtiddsgen

To create the example applications:

1. Set up the environment on your development machine: set the NDDSHOME environment variable and update your PATH as described in Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext DDS Getting Started Guide.
2. Create a directory to work in. In this example, we use a directory called **myhello**.
3. In the **myhello** directory, create a file called **HelloWorld.idl** that contains a user-defined data type:

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

4. Use *RTI Code Generator* (*rtiddsgen*) to generate sample code and a makefile. 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
   ```

   Edit the generated example code as described in Hands-On 1 of *Introduction to Publish/Subscribe*, in the RTI Connext DDS Getting Started Guide.

4.3.2 Building and Running an Application as a Kernel Task

There are two ways to build and run your *Connext DDS* application:

4.3.2.1 Using the Command Line

1. Set up your environment with the **wrenv.sh** script or **wrenv.bat** batch file in the VxWorks base directory. Execute the script with the **-p** parameter set to the correct version of VxWorks. For example:

   ```
   wrenv.sh -p vxworks-6.9
   ```

2. Set the NDDSHOME environment variable as described in Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext DDS Getting Started Guide.
3. Build the Publisher and Subscriber modules using the generated makefile. You may have to modify the HOST_TYPE, compiler and linker paths to match your development setup.
4. To use dynamic linking, remove the *Connext DDS* libraries from the link objects in the generated makefile.

(Note: steps 5-7 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 (you will see the prompt '->' in the shell) you can type **cmd** and then **help** for more information on how to load and run applications on your target.)

5. Launch Workbench.
6. Make sure your target is running VxWorks and is added to the Remote Systems panel. (To add a new target, click the **New Connection** button on the Remote System panel, select **Wind River VxWorks 6.x Target Server Connection**, click **Next**, enter the Target name or address, and click **Finish**).
7. Connect to the target and open a host shell by right-clicking the connected target in the **Target Tools** sub-menu.
8. In the shell:

   If you are using static linking: Load the **.so** file produced by the build:

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

   (Where 'directory' refers to the location of the generated object files.) If you are using dynamic linking: load the libraries first, in this order: **libnddscore.so**, **libnddsc.so**, **libnddscpp.so**; *then* load the **.so** file produced by the build.

   Note: If you are statically linking, and you try to load both the publisher and subscriber into the kernel, you will run into duplication of symbols due to the *Connext DDS* libraries being statically linked in both modules. To overcome that situation, see the "Notes for VxWorks 7 Platforms" section in the *RTI Connext DDS Core Libraries Platform Notes*, for an explanation about how to create a single Downloadable Kernel Module (DKM) containing both applications.
9. Run the **run_subscriber_application** or **run_publisher_application** function. For example:

   ```
   >taskSpawn "sub", 255, <floating_point_option>, 150000, run_subscriber_application, 38,
   10
   ```

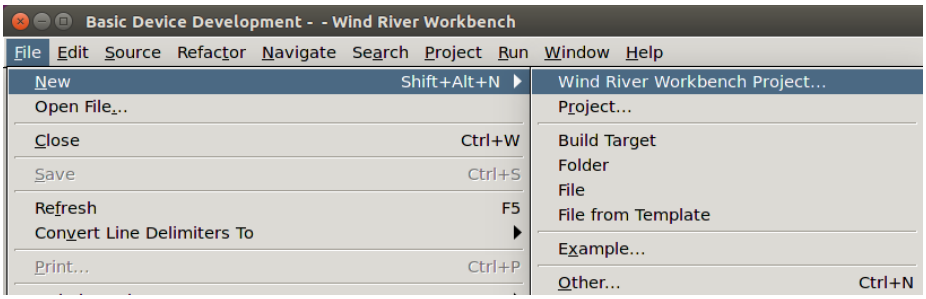
   Where <floating_point_option> is a numeric value that varies depending on the hardware. See Enabling Floating Point Coprocessor in Kernel Tasks, in the RTI Connext DDS Core Libraries Platform Notes.

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

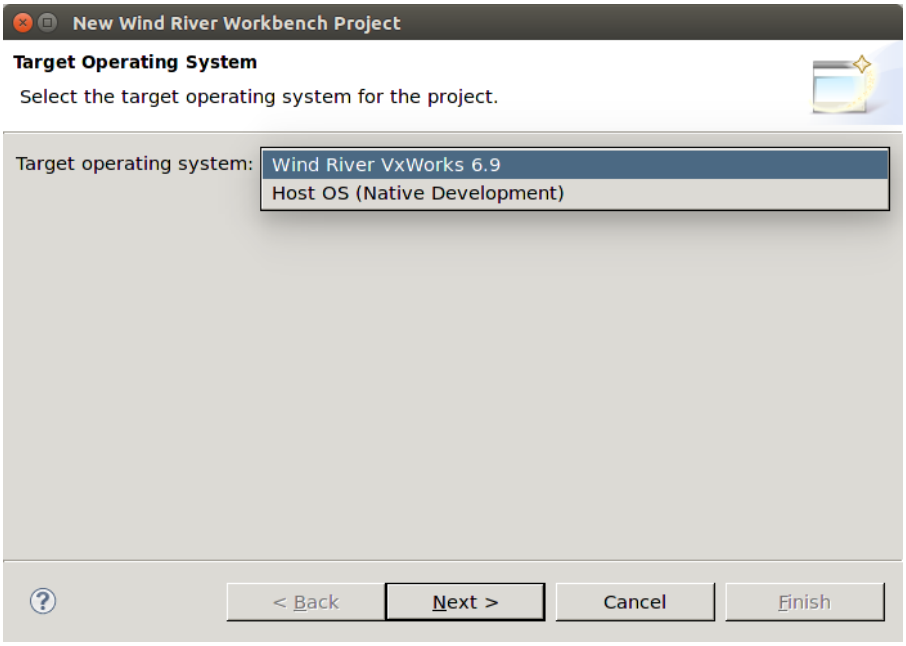
4.3.2.2 Using Workbench

Note: The following steps might vary slightly depending on your chosen version of VxWorks.

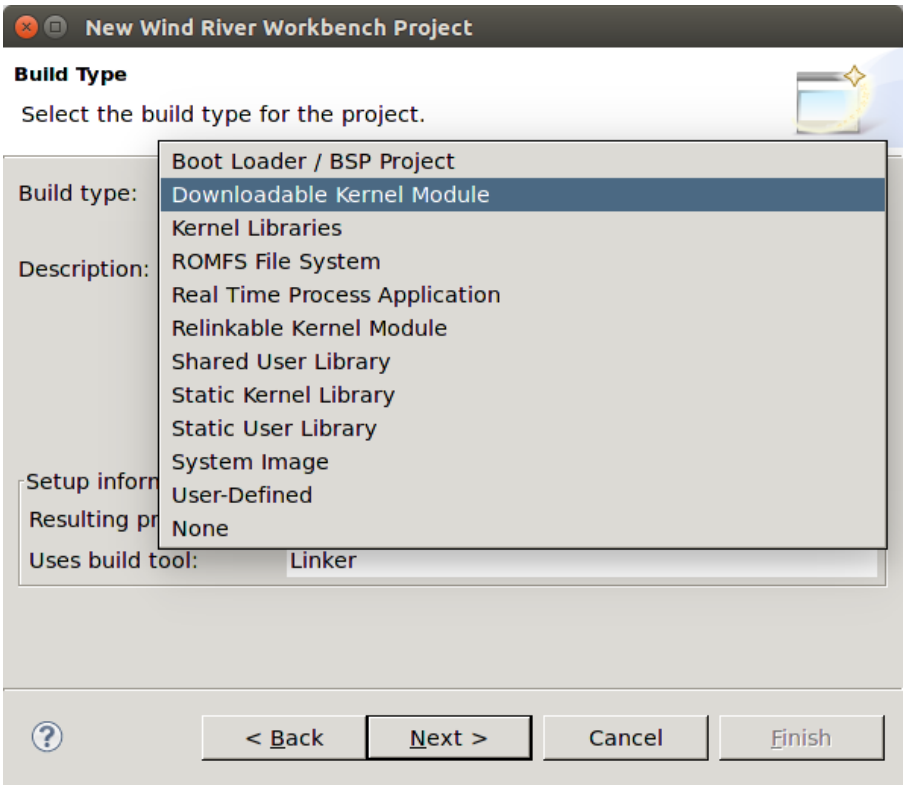
1. Start Workbench.
2. Select **File, New, Wind River Workbench Project**.



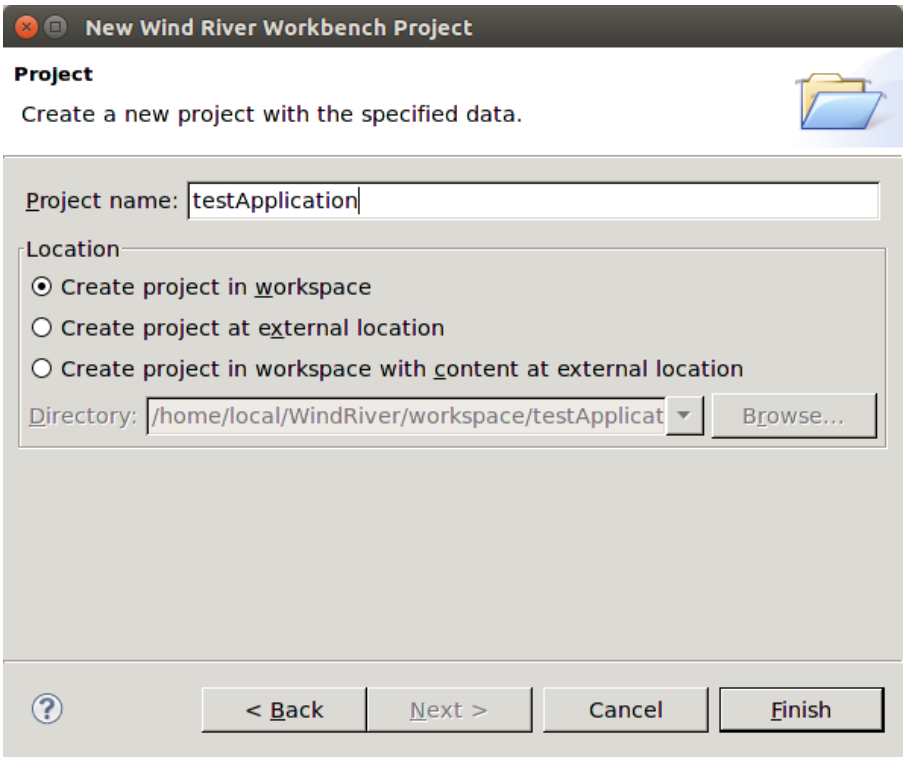
3. Select the desired **Target operating system**; click **Next**.



4. When prompted to choose a **Build type**, select **Downloadable Kernel Module**; click **Next**.

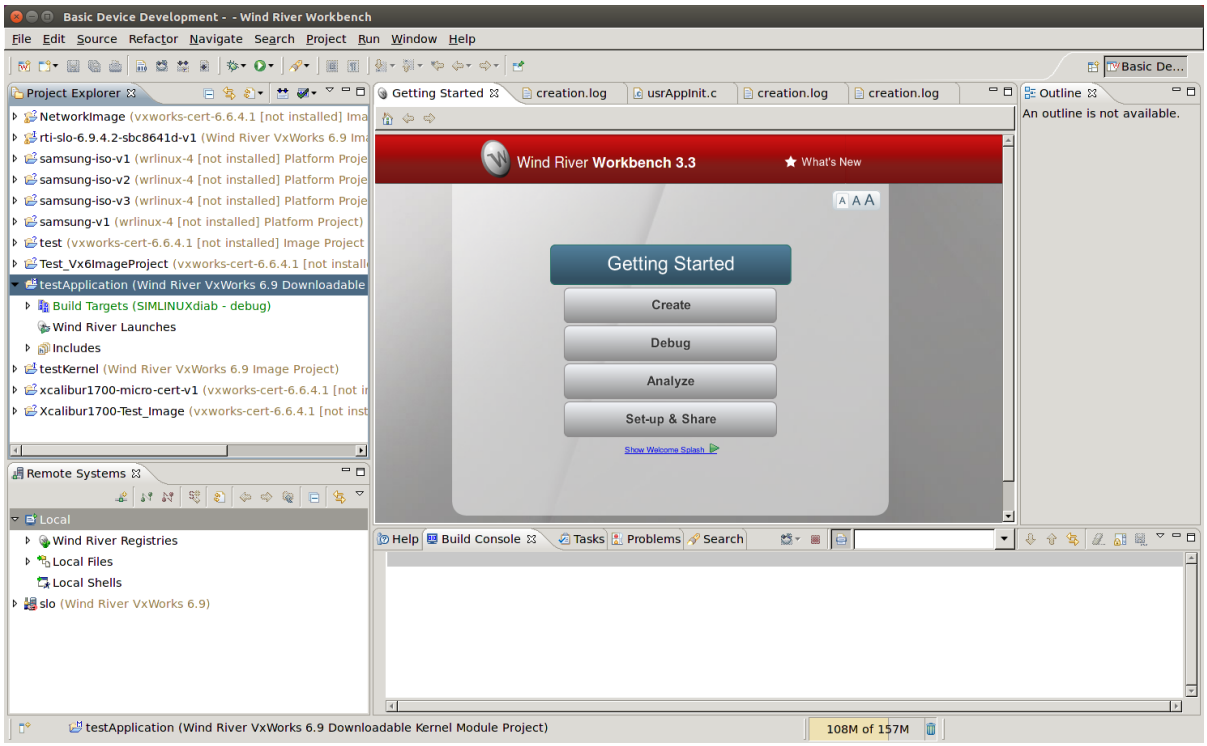


5. Give your project a name; click **Next**.

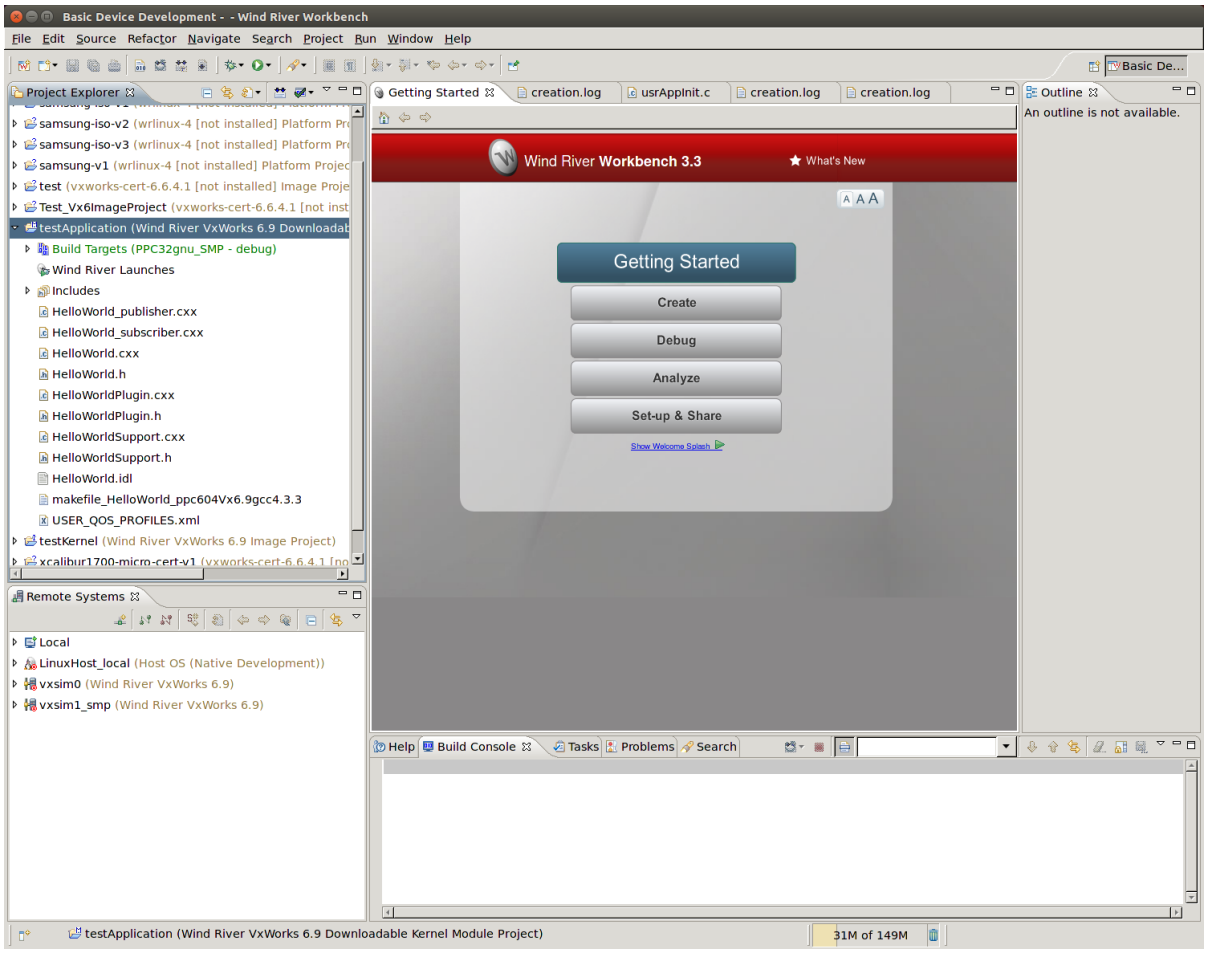


6. Leave everything else at its default setting; click **Finish**.

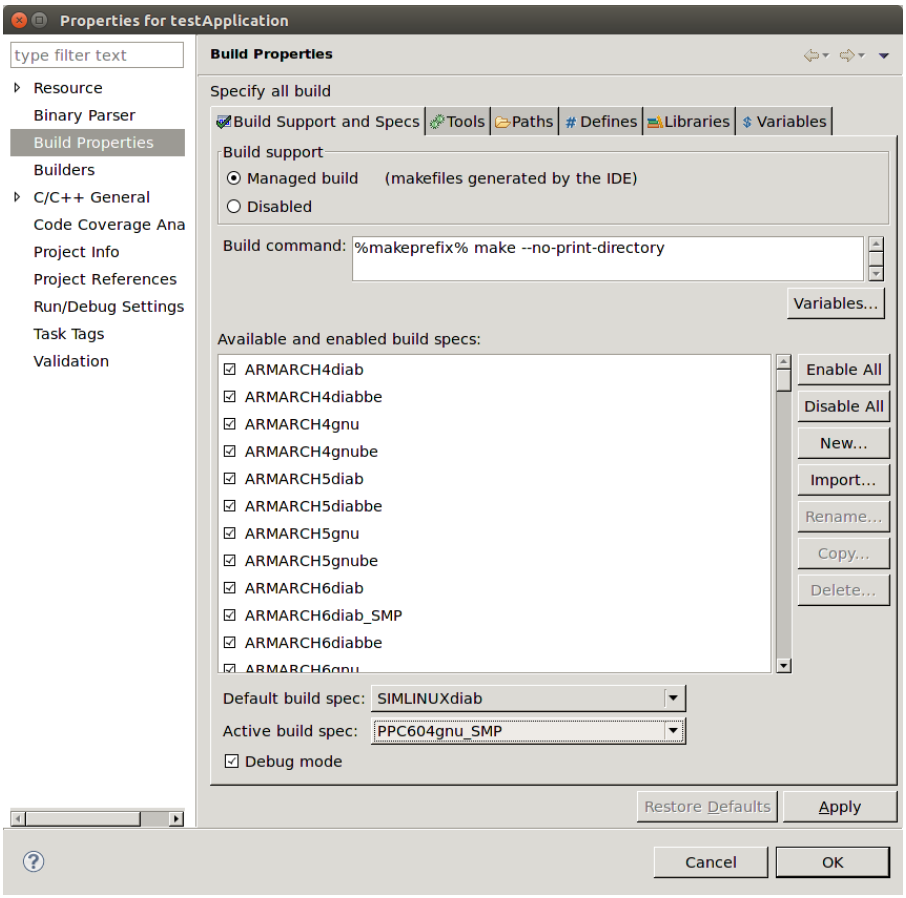
   Your project will be created at this time.



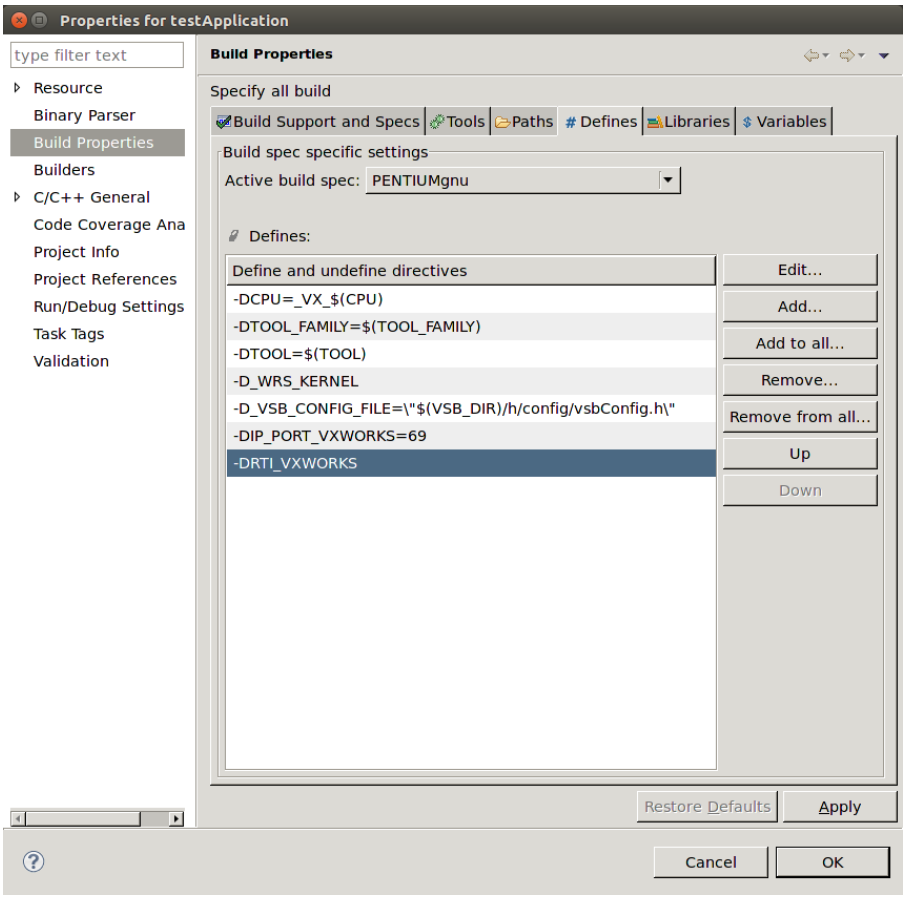
7. Copy the source and header files generated by *rtiddsgen* in 4.3.1 Generate Example Code and Makefile with rtiddsgen on page 18 into the project directory.
8. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.



9. Open the project Properties by right-clicking on the project in Project Explorer and selecting **Properties**.
10. In the dialog box that appears, select **Build Properties** in the navigation pane on the left.
11. In the **Build Support and Specs** tab, select the desired build spec from the **Active build spec** drop-down menu; click **Apply** to save the changes.



12. In the **Build Macros** or **Defines** tab, add **-DRTI_VXWORKS** to DEFINES in the Build macro definitions; click **Apply** to save the changes.



13. In the **Variables** tab, add to LIBPATH:

    -L/(NDDSHOME)/lib/<*architecture*>

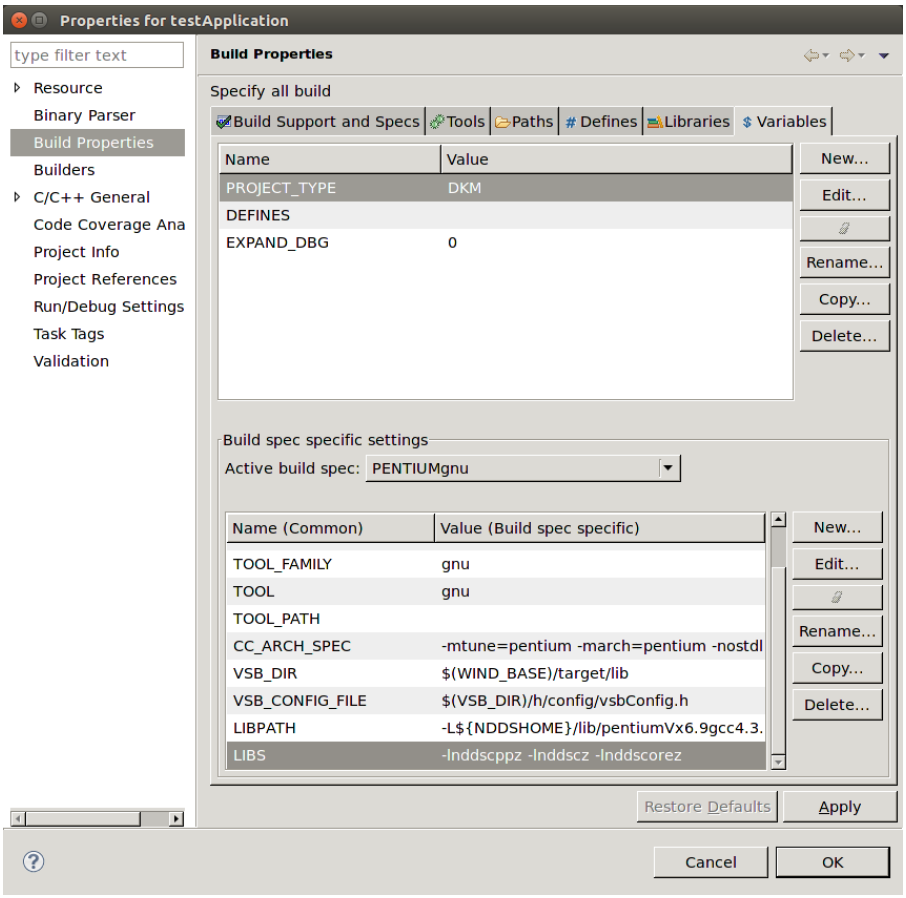
    If you are using *static* linking, add to LIBS:

    -lnddscppz -lnddscz -lnddscorez *(in that order)*

    If you are using *dynamic* linking, add to LIBS:

    -lnddscpp -lnddsc -lnddscore *(in that order)*

    Click **Apply** to save the changes.

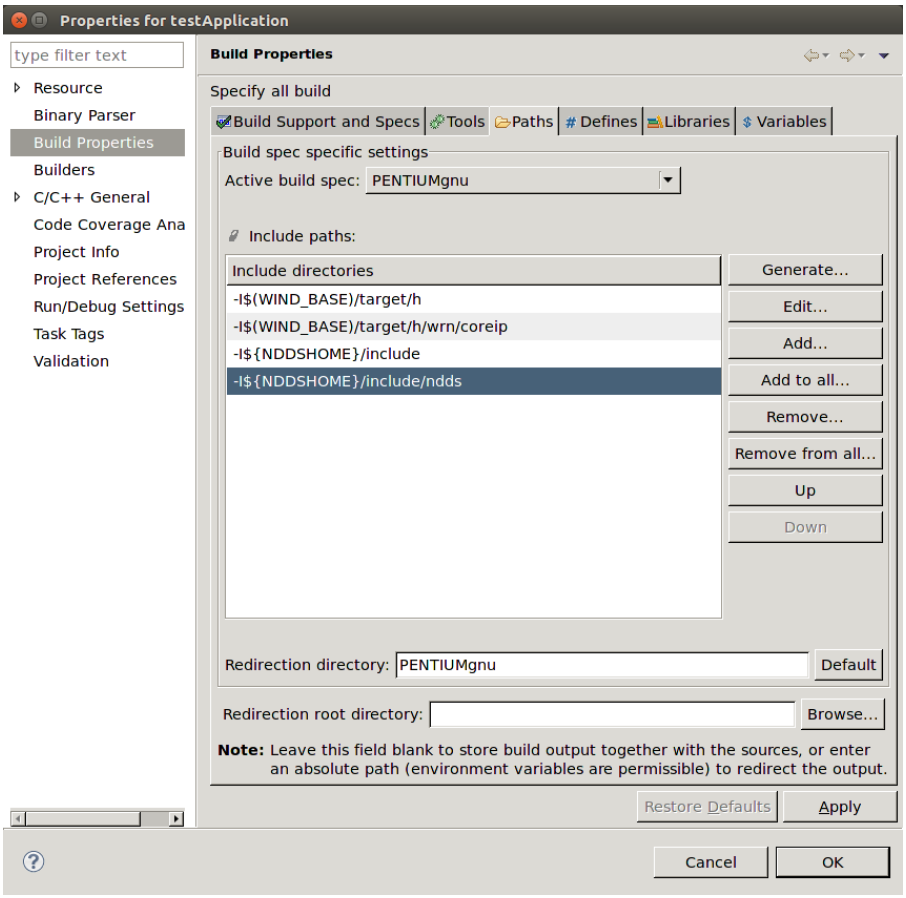


14. In the **Build Paths** or **Paths** tab, add both of these:

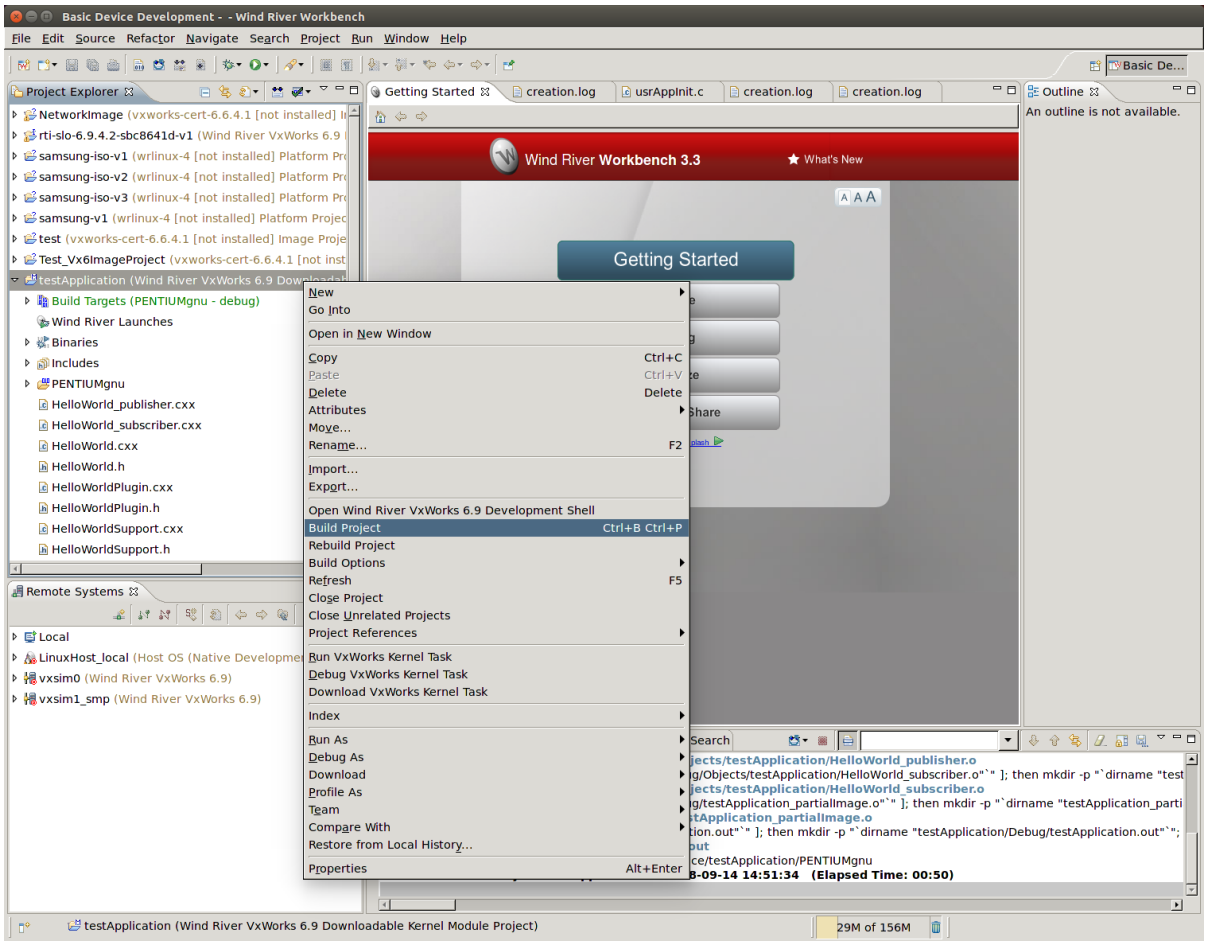
    -I$(NDDSHOME)/include

    -I$(NDDSHOME)/include/ndds

    Click **Apply** to save the changes.



15. Click **OK** to exit the Properties menu.
16. Build the project by right-clicking on the project in Project Explorer, then selecting **Build Project**.



17. Run the application as described starting in Step 5 in the 'Using the Command Line' section, except load **HelloWorld.out** instead of **HelloWorld_subscriber.so** when you get to Step 8.

4.3.3 Building and Running an Application as a Real-Time Process

There are two ways to build and run your *Connext DDS* RTP application:

- 4.3.3.1 Using the Command Line below
- 4.3.3.2 Using Workbench on page 31

4.3.3.1 Using the Command Line

1. Generate the source files and the makefile with *RTI Code Generator* (*rtiddsgen*).

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

   Please refer to the *RTI Code Generator User's Manual* for more information on how to use *rtiddsgen*.

2. Set up your environment with the **wrenv.sh** script or the **wrenv.bat** batch file in the VxWorks base directory. Execute the script with the **-p** parameter set to the correct version of VxWorks. For example:

```
wrenv.sh -p vxworks-6.9
```

3. Set the NDDSHOME environment variable as described in Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext DDS Getting Started Guide.

4. Build the Publisher and Subscriber modules using the generated makefile. You may need to modify the HOST_TYPE, compiler and linker paths to match your development setup.

   Notes:

   - Steps 5-12 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 (you will see a prompt '->' in the shell) you can type **cmd** and then **help** for more information on how to load and run applications on your target.)

     Using rtpSp:

     ```
     telnet raytheon-guy
     cd " <PROJECT ROOT FOLDER>"
     rtpSp "objs/<arch>/Foo_subscriber.vxe -domainId XX"
     ```

     Or using rtp exec:

     ```
     telnet raytheon-guy
     cd " <PROJECT ROOT FOLDER>"
     rtp exec objs/<arch>/Foo_subscriber.vxe - -domainId XX
     ```

   - If you want to dynamically link your RTP to the RTI libraries, make the following modifications the generated makefile:

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

5. 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.

6. Launch Workbench.

7. Make sure your target is running VxWorks.

8. 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** submenu.

9. Right-click on your target in the Target Manager window, then select **Run, Run RTP on Target**.

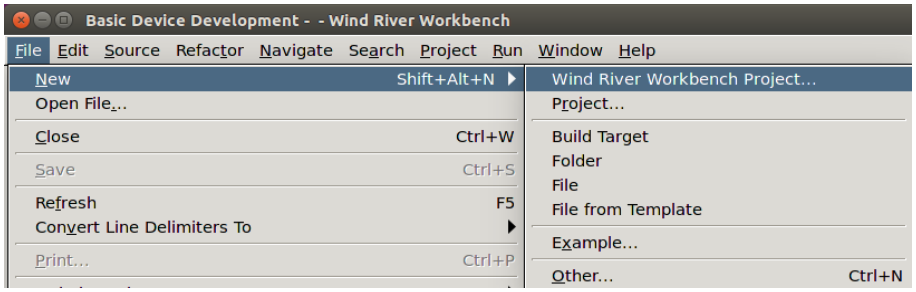
10. Set the **Exec Path on Target** to the **HelloWorld_subscriber.vxe** or the **HelloWorld_publisher.vxe** file created by the build.
11. Set the arguments (domain ID and number of samples, using -d <domain ID> and -s <number of samples>).

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

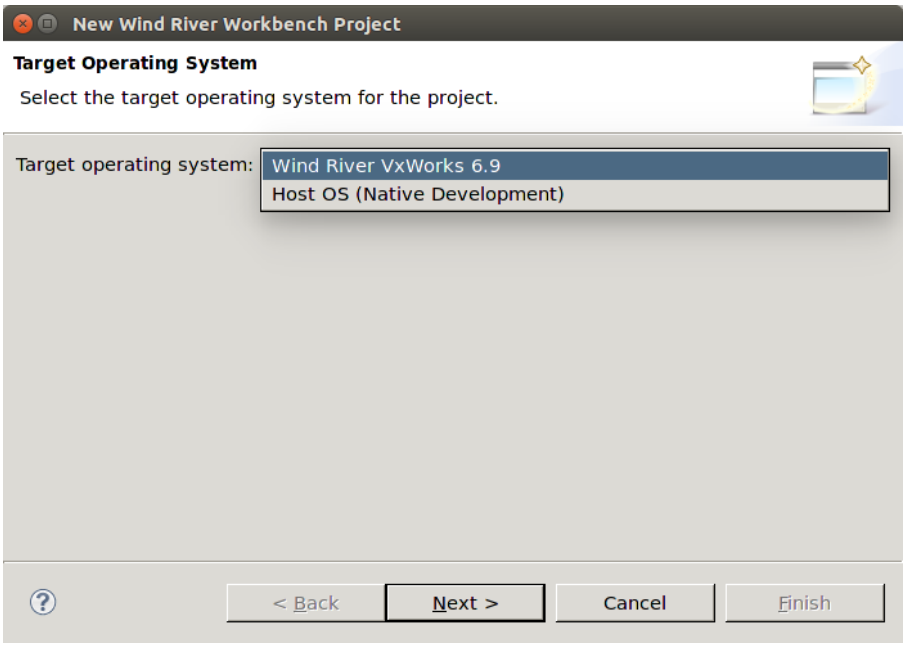
4.3.3.2 Using Workbench

Note: The following steps might vary slightly depending on your chosen version of VxWorks.

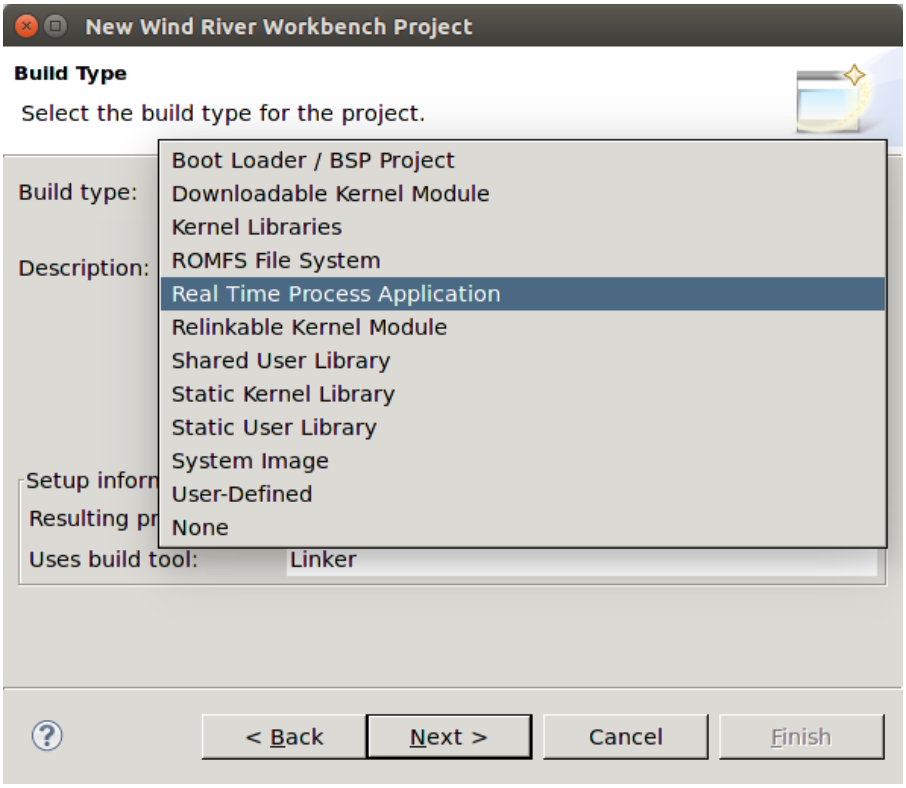
1. Start Workbench.
2. Select **File, New, Wind River Workbench Project**.



3. Select the desired **Target Operating System**; click **Next**.



4. When prompted to choose a **Build Type**, select **Real Time Process Application**; click **Next**.



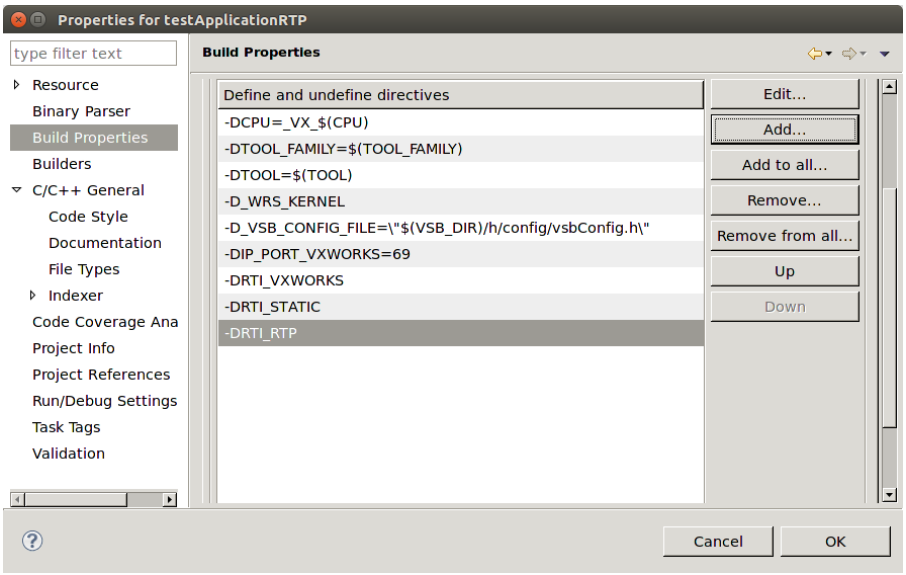
5. Give your project a name; click **Next**.
6. Leave everything else at its default setting; click **Finish**.

   Your project will be created at this time.
7. Copy the source and header files generated by *rtiddsgen* in 4.3.1 Generate Example Code and Makefile with rtiddsgen on page 18 into the project directory. There can only be one **main()** in your project, so you must choose *either* a subscriber or a publisher. If you want to run both, you will need to create two separate projects.
8. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.
9. Open the project Properties by right-clicking on the project in Project Explorer and selecting **Properties**.
10. In the dialog box that appears, select **Build Properties** in the navigation pane on the left.
11. In the **Build Support and Specs** tab, select the desired build spec from the **Active build spec** drop-down menu; click **Apply** to save the changes.
12. In the **Build Macros** or **Defines** tab, add the following to DEFINES in the Build macro definitions:

    -DRTI_VXWORKS

-DRTI_STATIC

-DRTI_RTP



13. In the **Variables** tab, add to LIBPATH:

    -L/(NDDSHOME)/lib/<*architecture*>

    If you are using *static* linking, add to LIBS:

    -lnddscppz -lnddscz -lnddscorez *(in that order)*

    If you are using *dynamic* linking, add to LIBS:

    -lnddscpp -lnddsc -lnddscore *(in that order)*

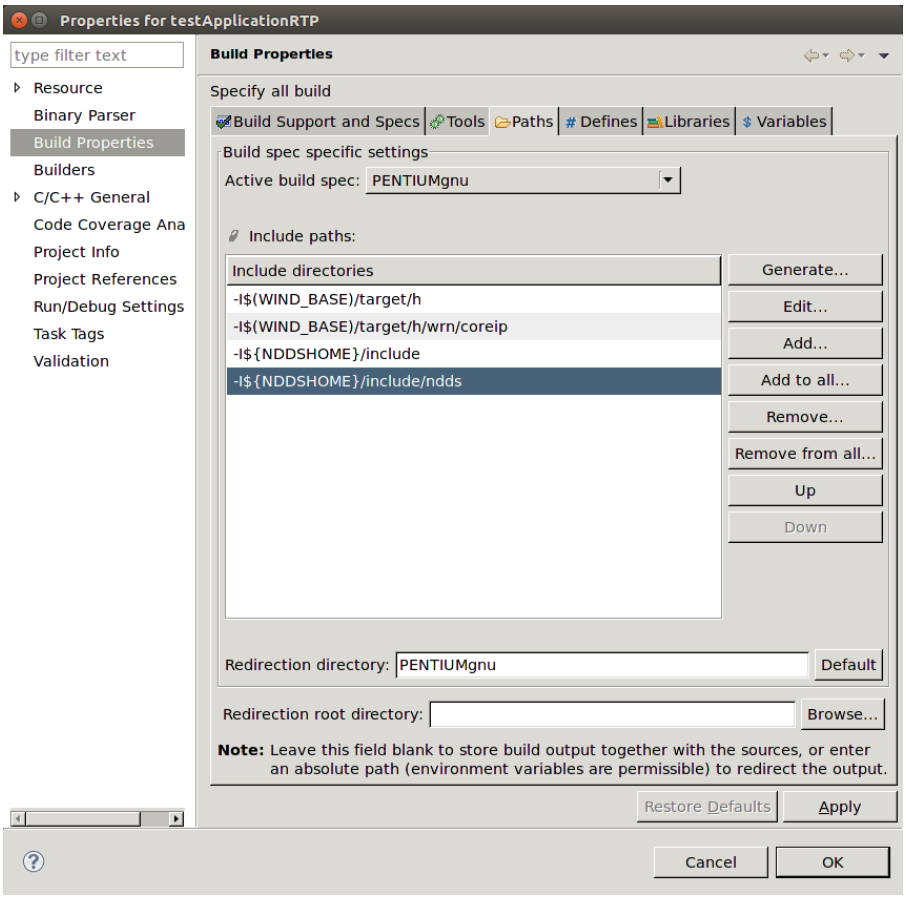
    Click **Apply** to save the changes.

14. In the **Build Paths** or **Paths** tab, add:

    -I$(NDDSHOME)/include

    -I$(NDDSHOME)/include/ndds

    Click **Apply** to save the changes.



15. Click **OK** to exit the Properties menu.
16. Build the project by right-clicking on the project in Project Explorer, then selecting **Build Project**.
17. Run the application as described starting in Step 5 in the Command Line section above.

4.4 Using DDS Ping and Spy

This section describes special usage notes when running the *RTI DDS Ping* and *Spy* command-line utilities on VxWorks systems. For complete details on using both utilities, see the API Reference HTML documentation (under Modules, Programming Tools).

RTI DDS Ping (*rtiddsping*) tests the connectivity of your system. It uses *RTI Connext DDS* to send and receive "Ping" messages to other *rtiddsping* applications running on the same or different computers.

RTI DDS Spy (*rtiddsspy*) shows you what is being published and subscribed to.

When running these utilities on VxWorks systems in RTP mode (as Real-Time processes):

- The utilities must be executed in a command prompt (running the "cmd" command in the C-shell)
- The utilities are statically linked so they don't require any LD_LIBRARY_PATH setup.

- The name of the utilities are suffixed with a "z" to indicate that they are statically linked (i.e. *DDS Ping* is called **rtiddspingz.vxe**).
- Each executable can be run as in any Linux OS (e.g., **rtiddspingz.vxe -help**).

When running these utilities on VxWorks systems in kernel mode (as DKMs):

- The modules **libnddscore.so**, **libnddsc.so**, and **libnddscpp.so** must first be loaded.
- After loading the *Connext DDS* modules, the utility module must be loaded in order to run it (i.e., **rtiddsping.so**).
- All the command-line options must be passed embedded in a single string (see examples below).
- The command must be typed in the VxWorks shell (either an rlogin shell, a target-server shell, or the serial line prompt).

The examples below illustrate how to run the utilities in Kernel mode. The string "vxworks prompt>" represents the prompt that the shell prints and is not part of the command that must be typed.

Ping:

```
vxworks prompt>  rtiddsping "-domainId 3 -publisher -numSamples 100"
vxworks prompt>  rtiddsping "-domainId 5 -subscriber -timeout 20"
vxworks prompt>  rtiddsping "-help"
```

Spy:

```
vxworks prompt>  rtiddsspy "-domainId 3 -topicRegex Alarm*"
vxworks prompt>  rtiddsspy "-help"
```

Or if the stack of the shell is not large enough, use "taskSpawn" to avoid overflowing the stack (each utility requires ~25 kB of stack).

Ping:

```
vxworks prompt> taskSpawn "rtiddsping", 100, <floating_point_option>, 50000, rtiddsping, \
   "-domainId 3 -publisher -numSamples 100"
vxworks prompt> taskSpawn "rtiddsping", 100, <floating_point_option>, 50000, rtiddsping, \
  "-domainId 5 -subscriber -timeout 20"
vxworks prompt> taskSpawn "rtiddsping", 100, <floating_point_option>, 50000, rtiddsping, "-
help"
```

Spy:

```
vxworks prompt> taskSpawn "rtiddsspy", 100, <floating_point_option>, 50000, rtiddsspy, \
  "-domainId 3 -topicRegex Alarm*"
vxworks prompt> taskSpawn "rtiddsspy", 100, <floating_point_option>, 50000, rtiddsspy, "-help"
```

Where <floating_point_option> is a numeric value that varies depending on the hardware. See Enabling Floating Point Coprocessor in Kernel Tasks, in the RTI Connext DDS Core Libraries Platform Notes.

Chapter 5 Getting Started on Wind River Linux Systems

This section provides instructions on building and running *Connext DDS* applications on a Wind River Linux system.

It will guide you through the process of compiling and running the Hello World application on a Wind River Linux system.

In the following steps:

- Steps 1-5 must be executed on the host machine in a shell that has all the required environment variables. For details, see Set Up Environment Variables (rtisetenv), in "Hands-On 1" of *Introduction to Publish/Subscribe*, in the RTI Connext DDS 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 'ppc85xxWRLinux2.6gcc4.3.2'.
- We assume that you have **make** installed. If you have **make**, you can use the generated makefile to compile. If you do not have **make**, 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.)

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 generate sample code and a makefile. Choose either C or C++.

   For C:

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

   For C++:

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

   Edit the example code to add the line **sprintf(instance->msg, "Hello World! (%d)", count);** as follows:

   ```
   for (count=0; (sample_count == 0) || (count < sample_count); ++count) {

       printf("Writing HelloWorld, count %d\n", count);

       /* Modify the data to be written here */
       sprintf(instance->msg, "Hello World! (%d)", count);

       /* Write data */
       retcode = HelloWorldDataWriter_write(
           HelloWorld_writer, instance, &instance_handle);
       if (retcode != DDS_RETCODE_OK) {
           fprintf(stderr, "write error %d\n", retcode);
       }

       NDDS_Utility_sleep(&send_period);

   }
   ```

4. Set up your environment with the **wrenv.sh** script in the Wind River Linux base directory.

   ```
   wrenv.sh -p <WRLINUX_PACKAGE>
   ```

   where <*WRLINUX_PACKAGE*> is the package you use in the installation (for example, **wrlinux-3.0**).

5. With the NDDSHOME environment variable set, build the Publisher and Subscriber modules using the generated makefile.

   ```
   make -f makefile_HelloWorld_<architecture>
   ```

   After compiling, you will find the application executables in **myhello/objs/<***architecture***>**.

6. Connect to the Wind River Linux target (using telnet, ssh, serial console, connection manager, etc.) and start the subscriber application, **HelloWorld_subscriber**.

```
HelloWorld_subscriber
```

In this shell, in C, 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...
```

In C++:

```
No data after 1 second
No data after 1 second
No data after 1 second
```

7. Connect to the Wind River Linux target and start the publisher application, **HelloWorld_publishe**r.

```
HelloWorld_publisher
```

In this second (publishing) shell, you should see:

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

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

In C:

```
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...
```

In C++:

```
Received data
   msg: "Hello World! (0)"
Received data
   msg: "Hello World! (1)"
Received data
   msg: "Hello World! (2)"
```