RTI Connext DDS Core Libraries

Getting Started Guide

Addendum for Embedded Systems

Version 6.0.1

© 2020 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
March 2020.

Trademarks

RTI, Real-Time Innovations, Connext, NDDS, the RTI logo, 1RTI and the phrase, "Your Systems. Working as one," are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

This is an independent publication and is neither affiliated with, nor authorized, sponsored, or approved by, Microsoft Corporation.

The security features of this product include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

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 Embedded UNIX-like Systems

This document provides instructions on building and running *Connext DDS* 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 Building and Running Hello World, in the RTI Connext DDS Core Libraries 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 Step 1, Set up the Environment, in the RTI Connext DDS Core Libraries 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 'i86Lynx4.0.0gcc3.2.2'.
- 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 DDS Types Defined at Compile Time, 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.idlgmake -f makefile_
   HelloWorld_<architecture>
   gmake -f makefile_HelloWorld_<architecture> HelloWorldSubscribergmake -f makefile_
   HelloWorld_<architecture> HelloWorldPublisher
   ```

   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, in the RTI Connext DDS Core Libraries Getting Started Guide.

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

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 *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 as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries 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 the *RTI Connext DDS Core Libraries 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 RTI Code Generator, in the RTI Connext DDS Core Libraries Getting Started Guide.

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 'reset'.

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 12

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 17

For tips on using RTI DDS Ping and Spy, see 4.4 Using DDS Ping and Spy on page 32.

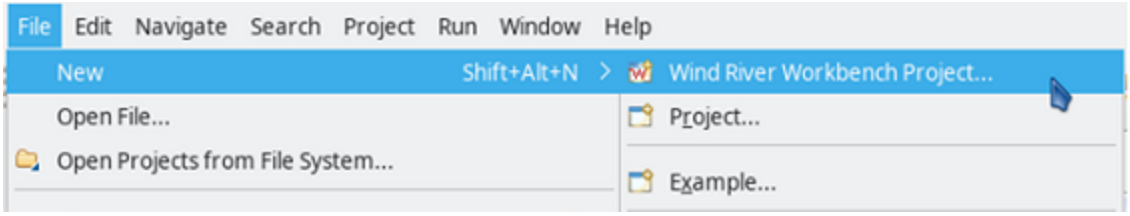
4.1 Building the VSB

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.

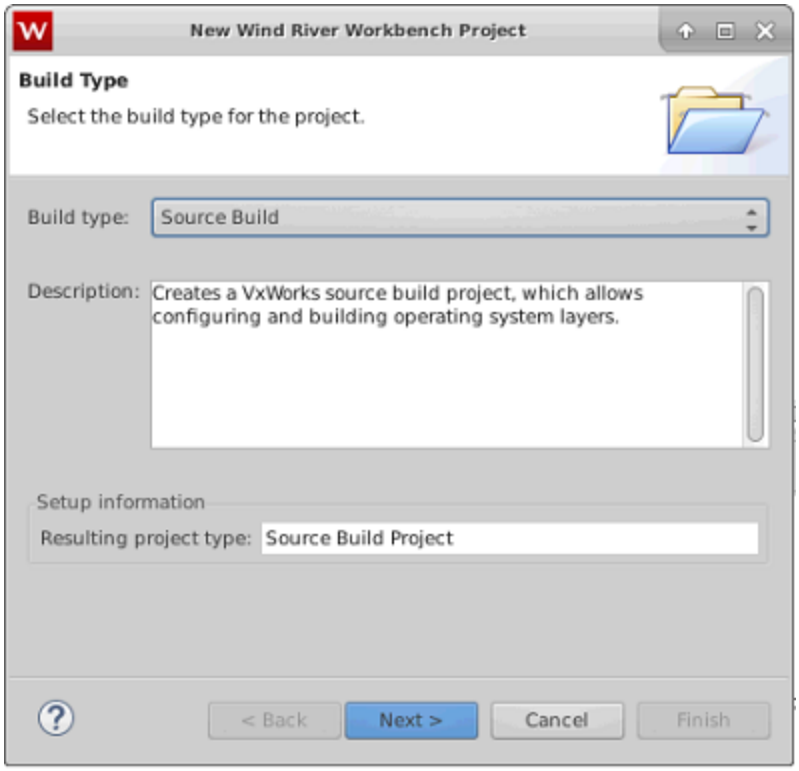
The following steps use the VSB defaults. For further information and special customizations, please refer to Wind River's documentation:
https://docs.windriver.com/bundle/Configuration_and_Build_Guide_Edition_9_1/page/1597954.html

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.

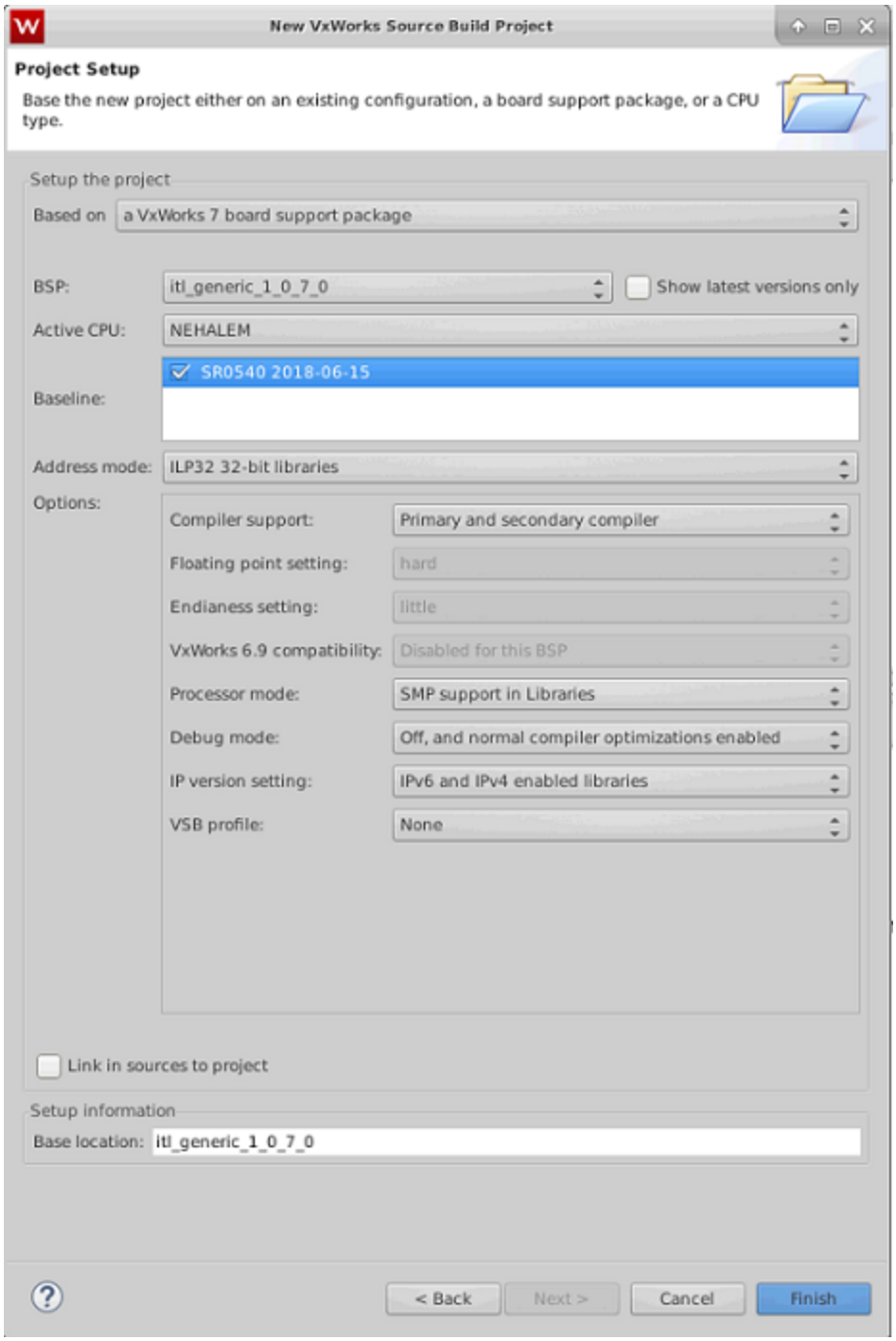
1. Launch Workbench.
2. Select **File, New, Wind River Workbench Project**.



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.

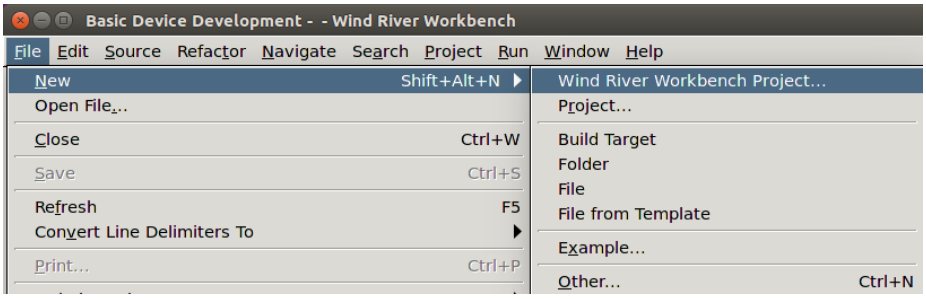
4.2 Building the Kernel

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.

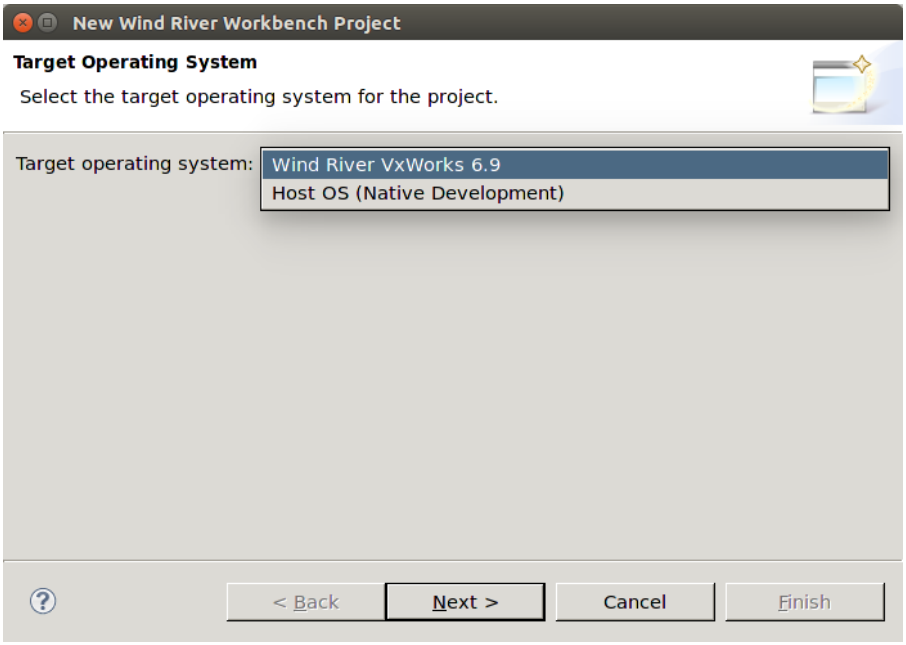
Before you start, you should be familiar with building and deploying a default working kernel on your target.

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

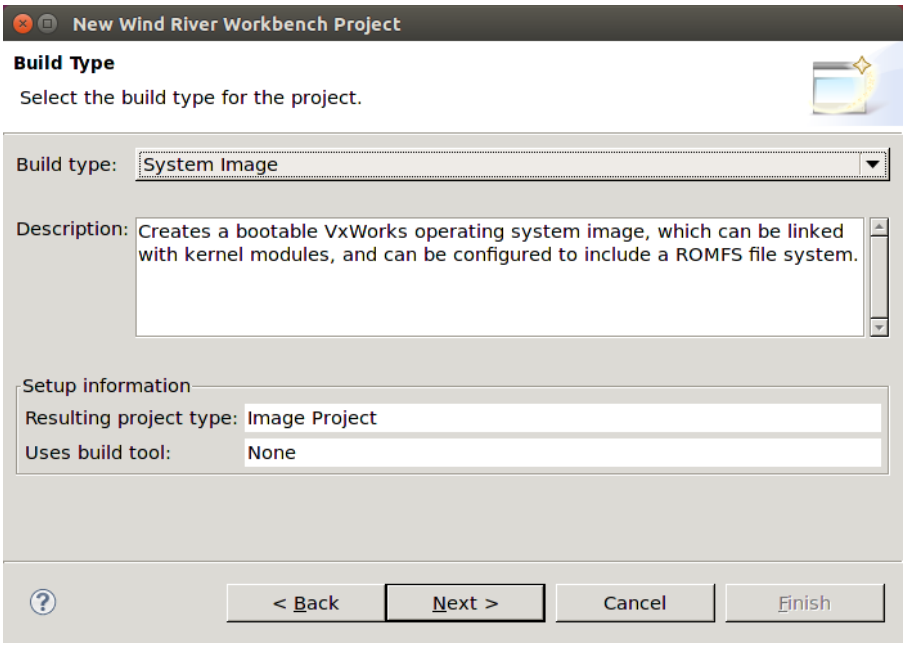
1. Launch Workbench.
2. Select **File, New, Wind River Workbench Project**.



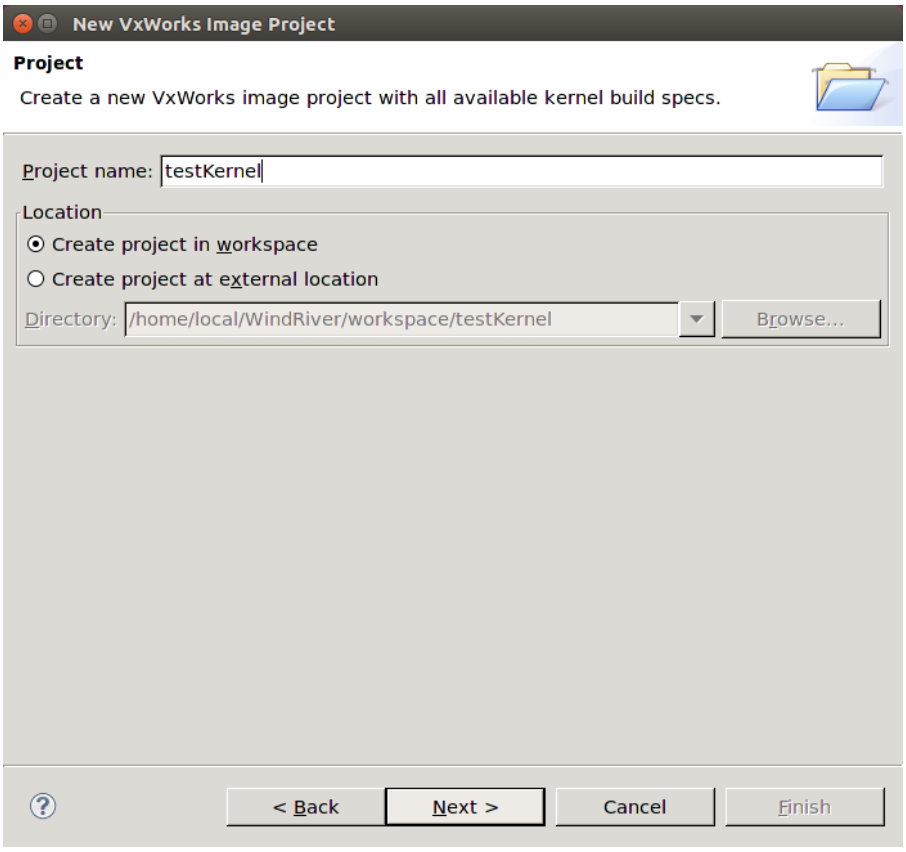
3. Select the desired **Target operating system**; click **Next**.



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**.



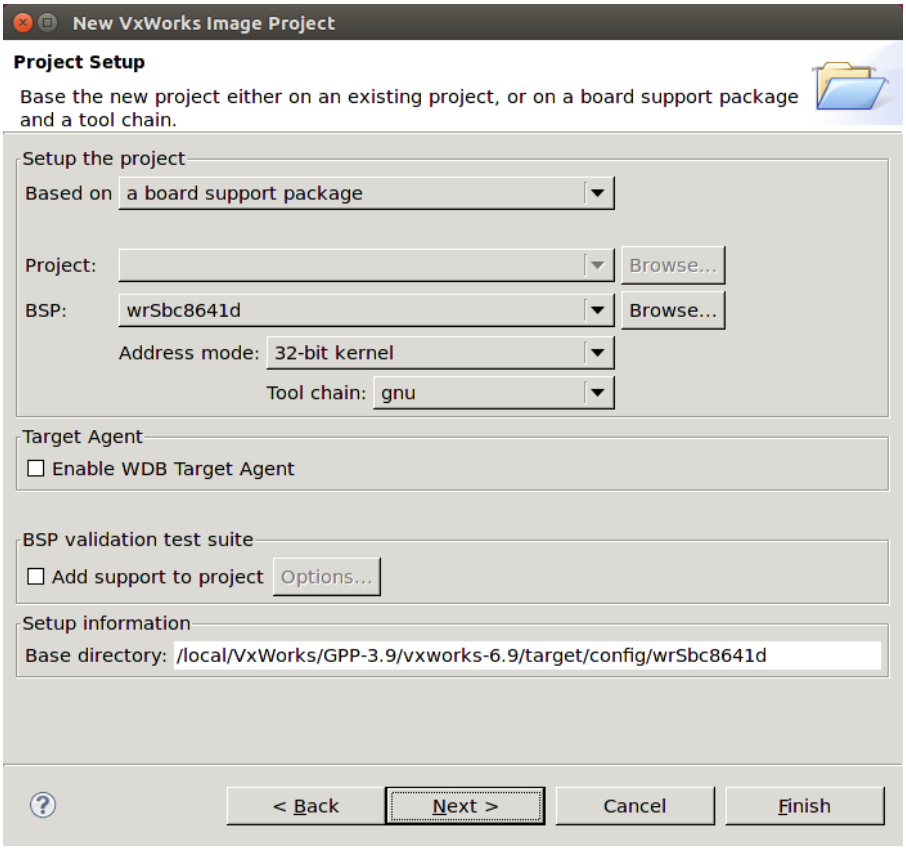
5. Give your project a name; click **Next**.



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

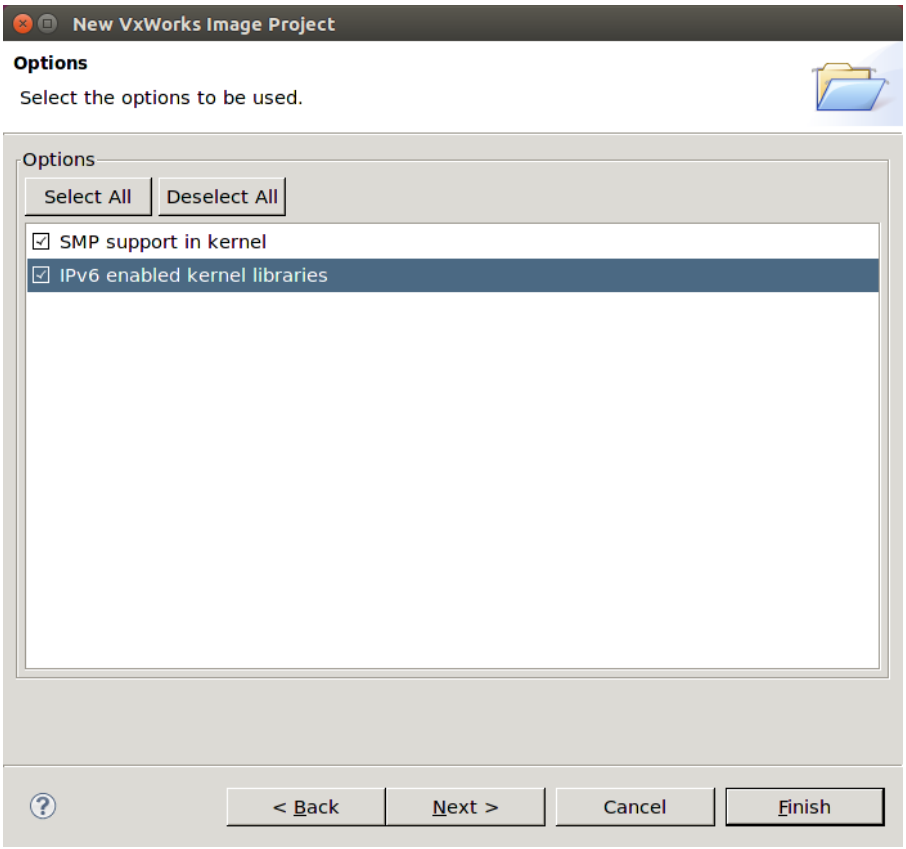
   VxWorks 7: In Project Setup, for the **Based on** field, choose **a source build project**. For the **Project**, choose the VSB you created and built in the previous section. The BSP, SMP support and other options will be correctly populated from the VSB configuration.

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

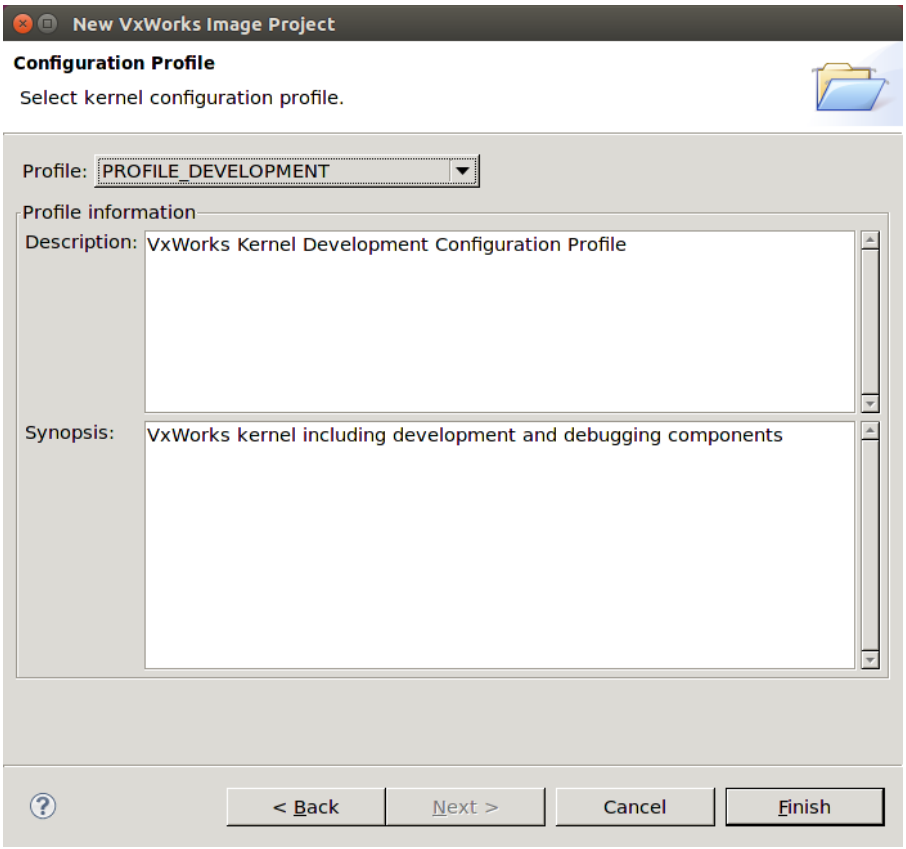


3. In **Options**, select **SMP support in kernel** if your BSP supports it and you want to enable symmetric multi-processing capability in the kernel.

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



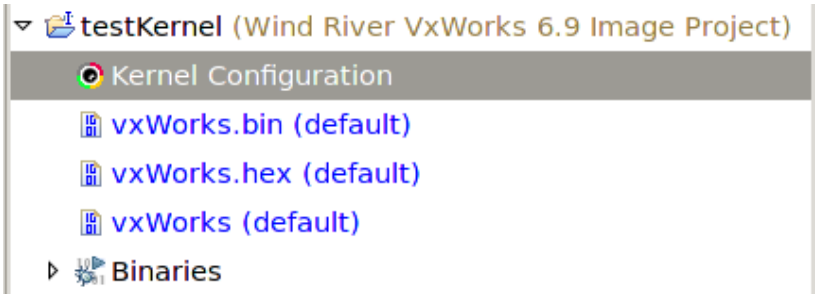
4. Optionally, select a configuration profile from the drop-down menu.



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

   Your project will be created at this time.

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



7. Add **Operating System Components, Kernel Components, _thread variables support**.

8. Make sure you have the following components enabled: INCLUDE_TIMESTAMP, INCLUDE_SHARED_DATA.

   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*.

9. If you plan to use the Request/Reply C++ API in kernel mode, you will need the following components: FOLDER_CPLUS, FOLDER_CPLUS_STDLIB, and CPLUS_LANG.

   If you plan to use the conventional *Connext DDS* C++ API, but not the Request/Reply C++ API , you can forego the STL includes, as well as the exceptions support, provided you don't use those C++ features in your application.
10. 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.
11. 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).
12. 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.
13. 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 Step 1, Set up the Environment, in the RTI Connext DDS Core Libraries 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 as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries 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
   ```

   Edit the generated example code as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries 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 Step 1, Set up the Environment, in the RTI Connext DDS Core Libraries 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.
9. Run the **subscriber_main** or **publisher_main** function. For example:

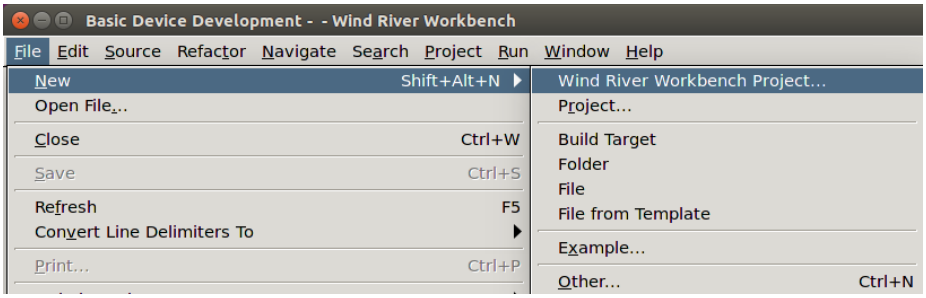
   ```
   >taskSpawn "sub", 255, 0x8, 150000, subscriber_main, 38, 10
   ```

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

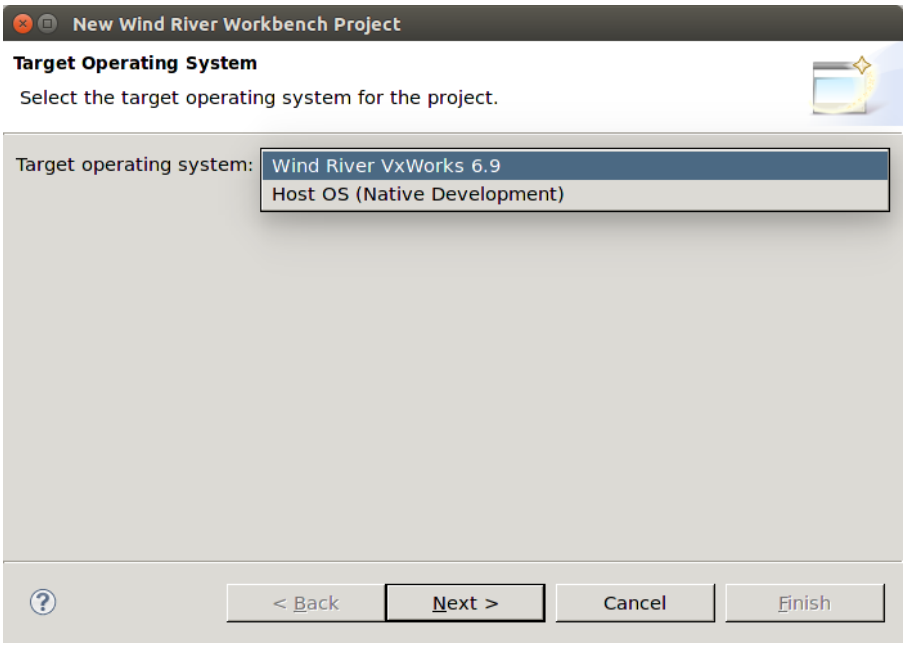
4.3.2.2 Using Workbench

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

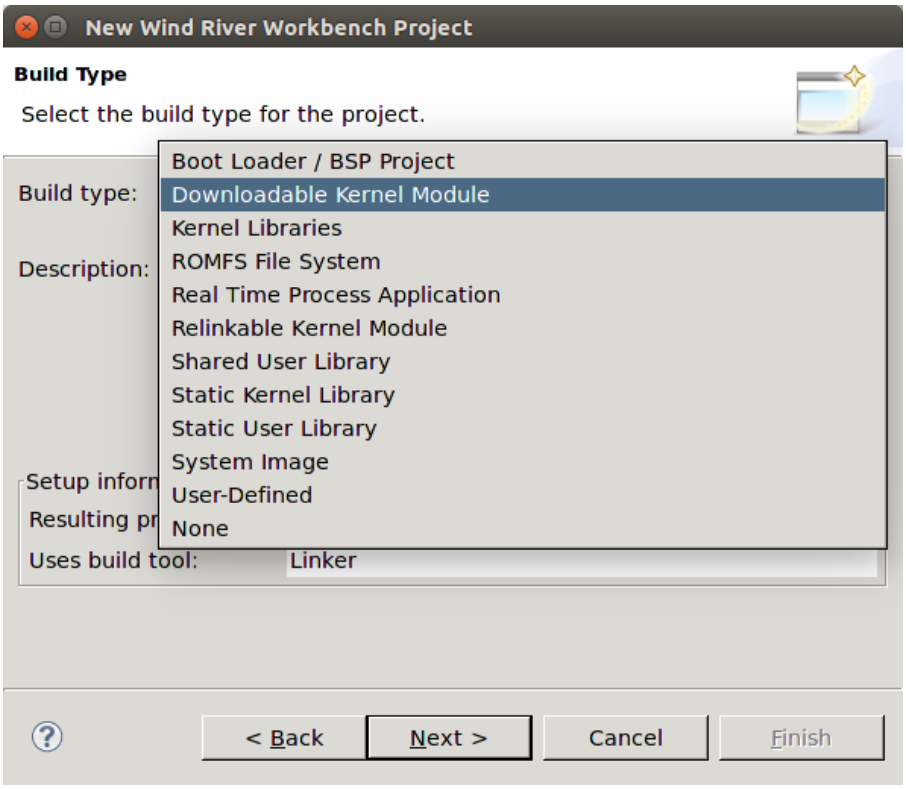
1. Start Workbench.
2. Select **File, New, Wind River Workbench Project**.



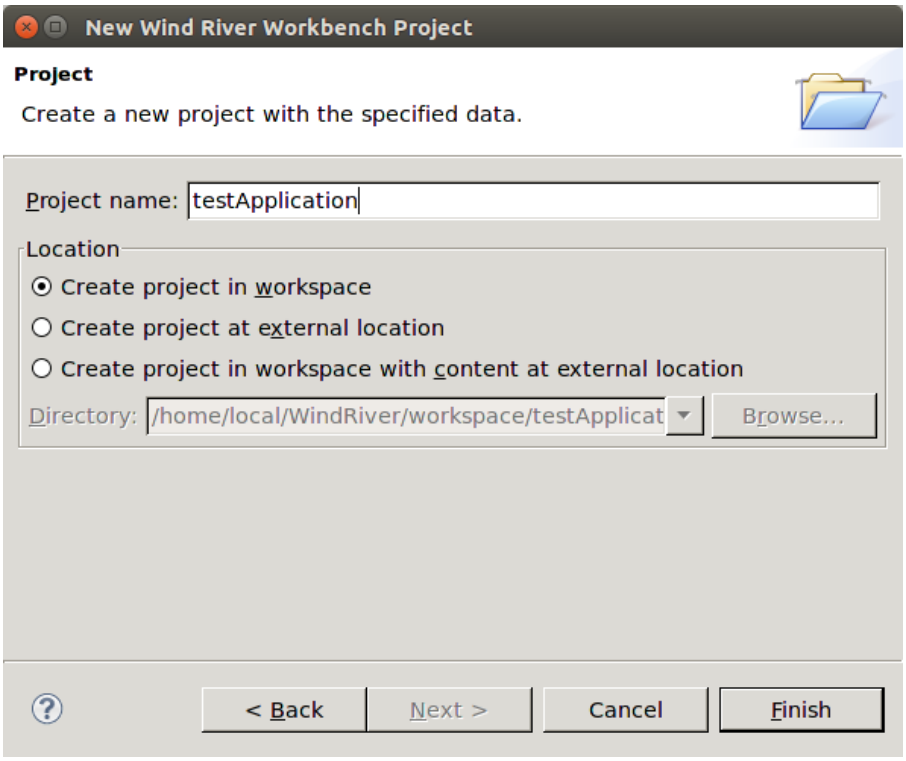
3. Select the desired **Target operating system**; click **Next**.



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

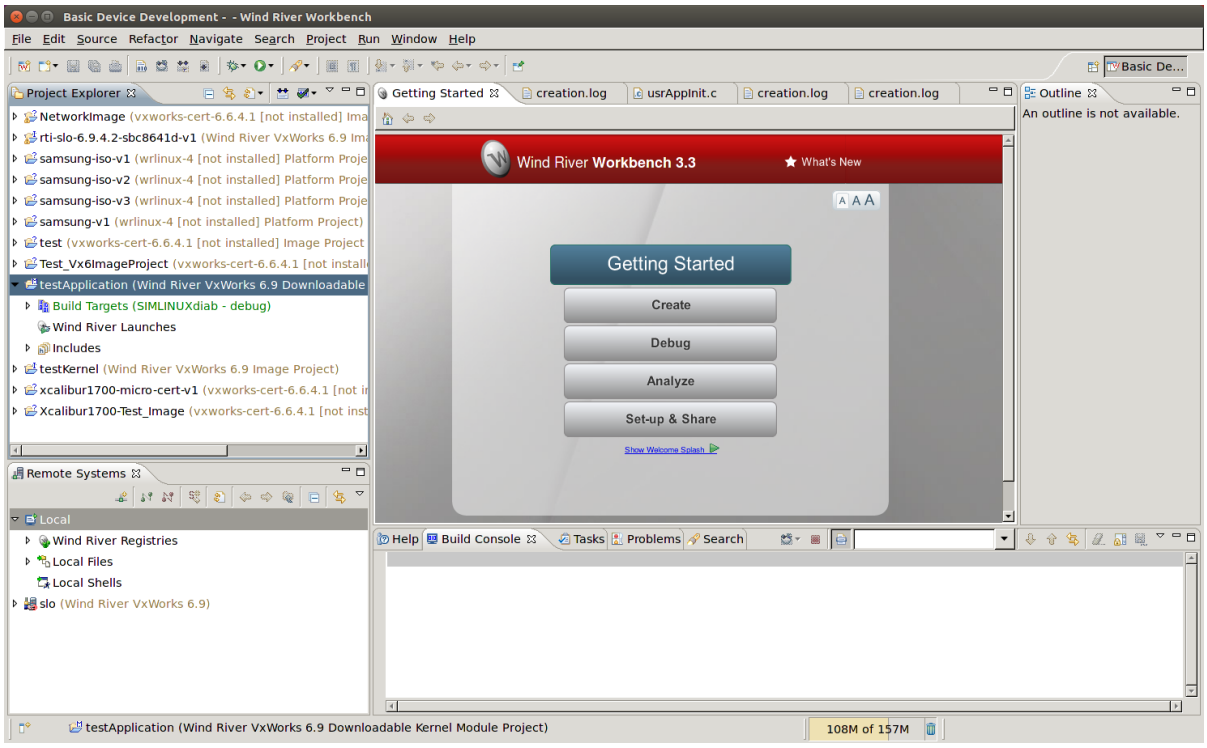


5. Give your project a name; click **Next**.

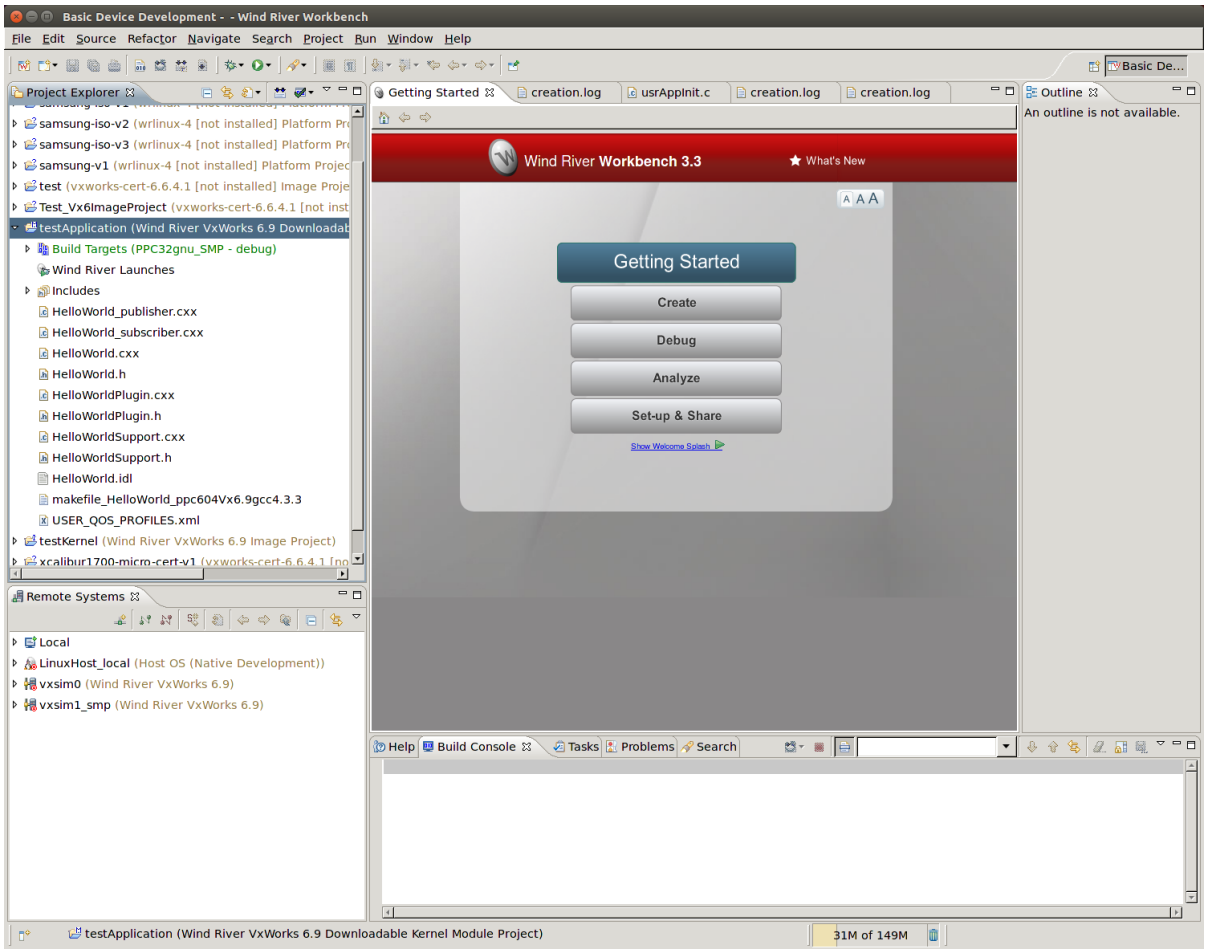


6. Leave everything else at its default setting; click **Finish**.

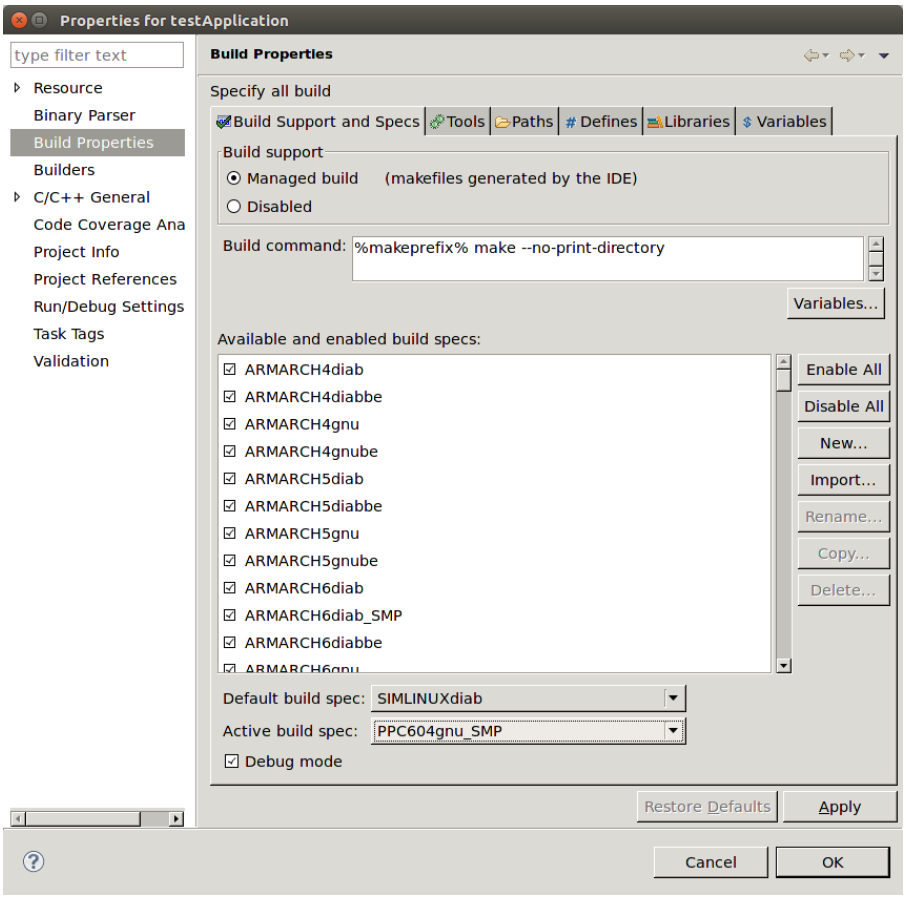
   Your project will be created at this time.



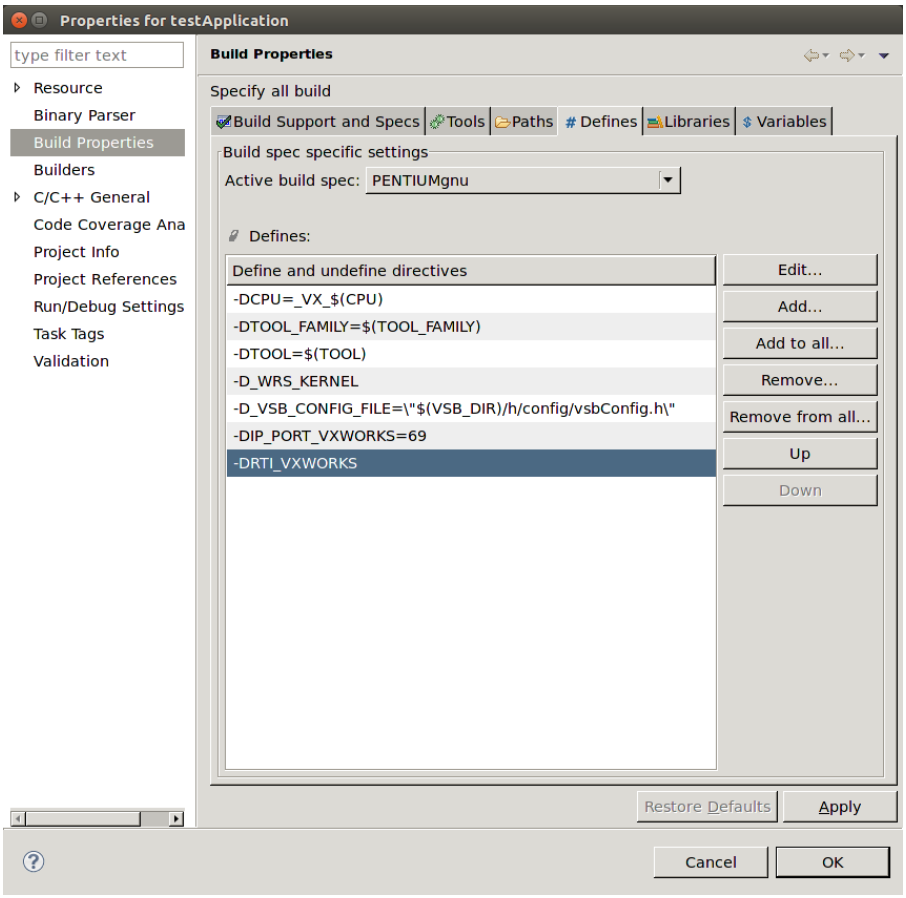
7. Copy the source and header files generated by *rtiddsgen* in4.3.1 Generate Example Code and Makefile with rtiddsgen on page 17 into the project directory.
8. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.



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*>

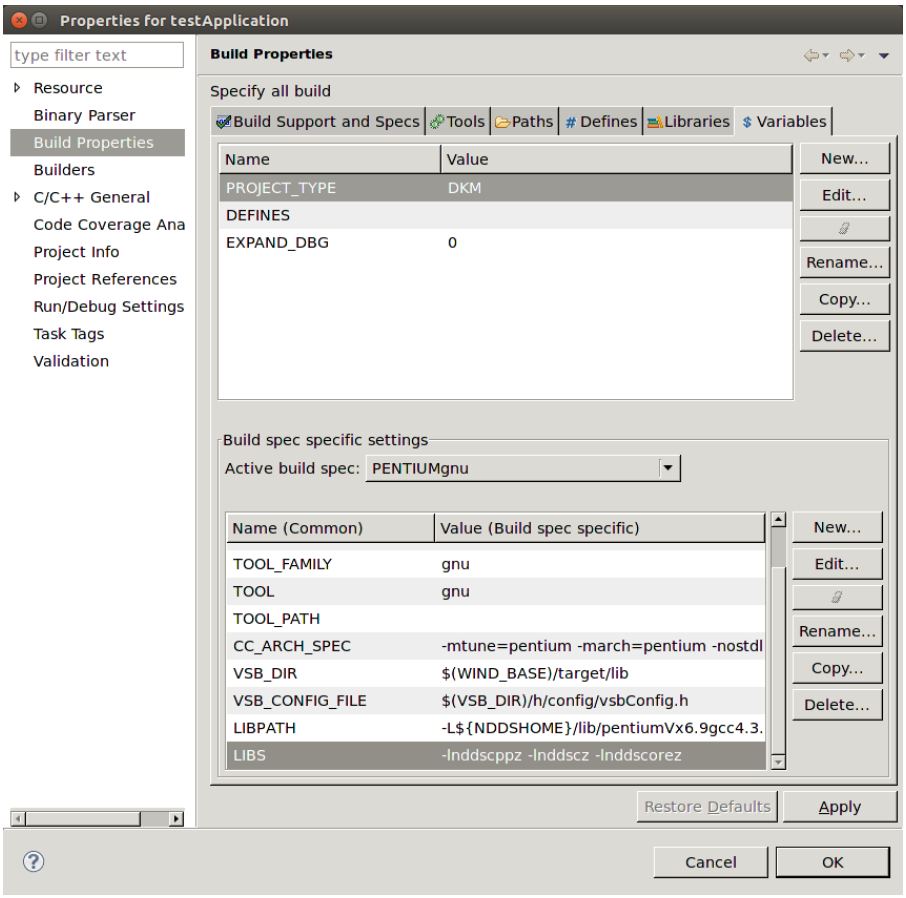
    If you are using *static* linking, add to LIBS:

    -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.

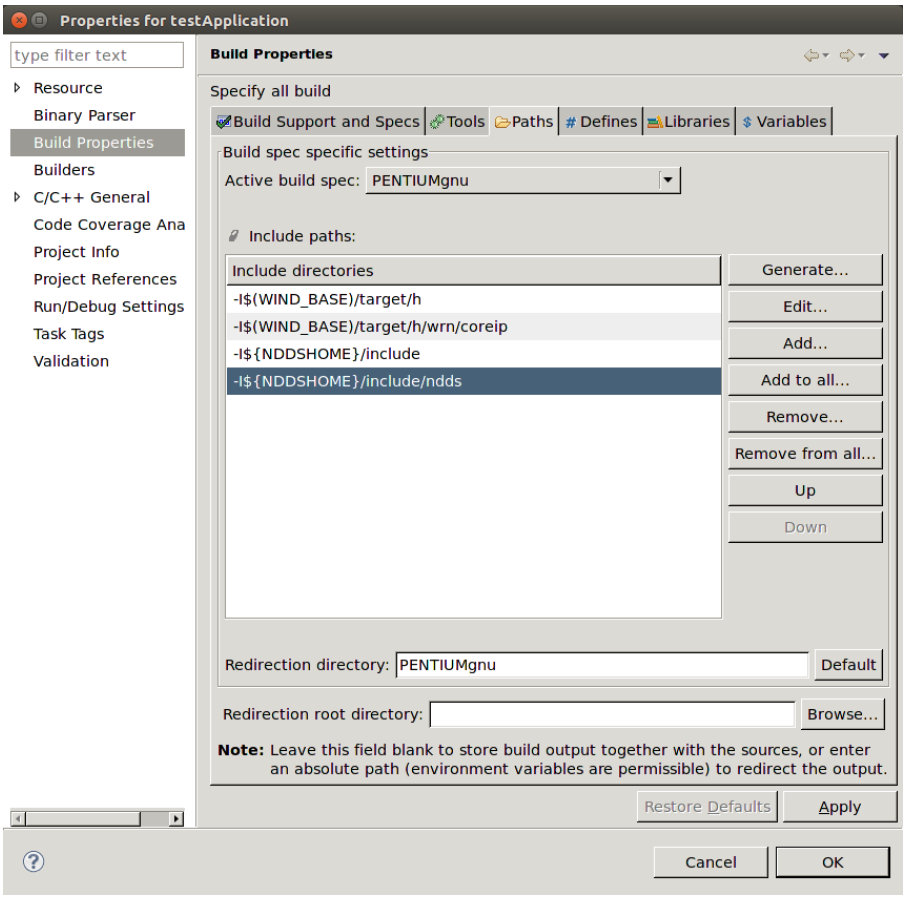


14. In the **Build Paths** or **Paths** tab, add both of these:

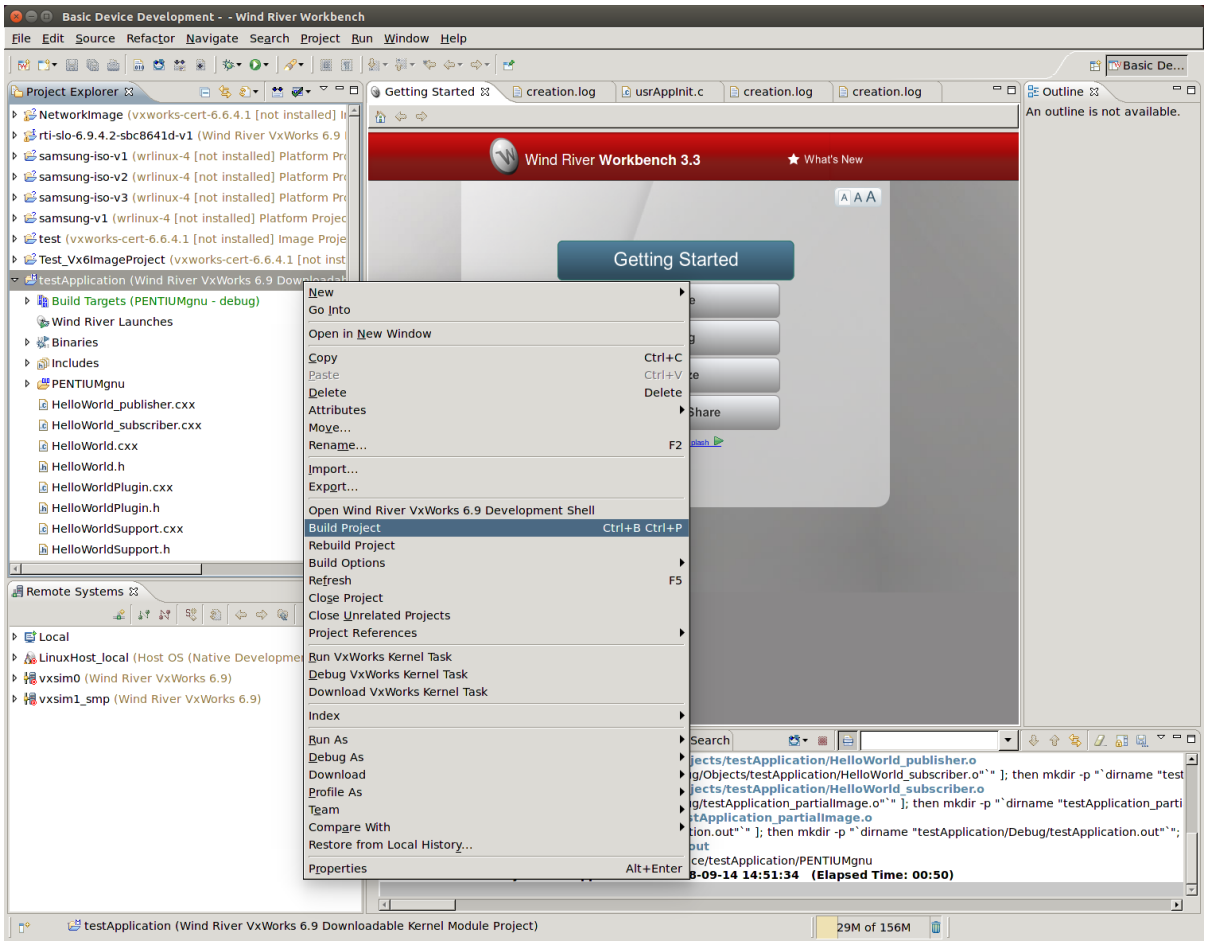
    -I$(NDDSHOME)/include

    -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 the next page

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 Step 1, Set up the Environment, in the RTI Connext DDS Core Libraries 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.)
   - 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>)
```

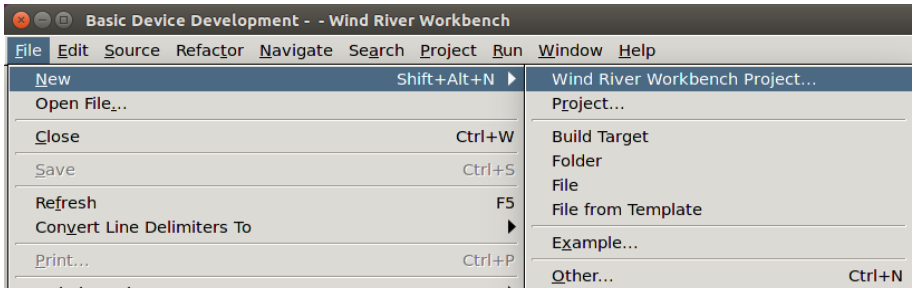
5. Add to the **LD_LIBRARY_PATH** environment variable the path to your RTI libraries as well as the path to **libc.so.1** of your VxWorks installation to launch your RTP successfully.
6. Launch Workbench.
7. Make sure your target is running VxWorks.
8. Connect to the target with the target manager and 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 a space separator).

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

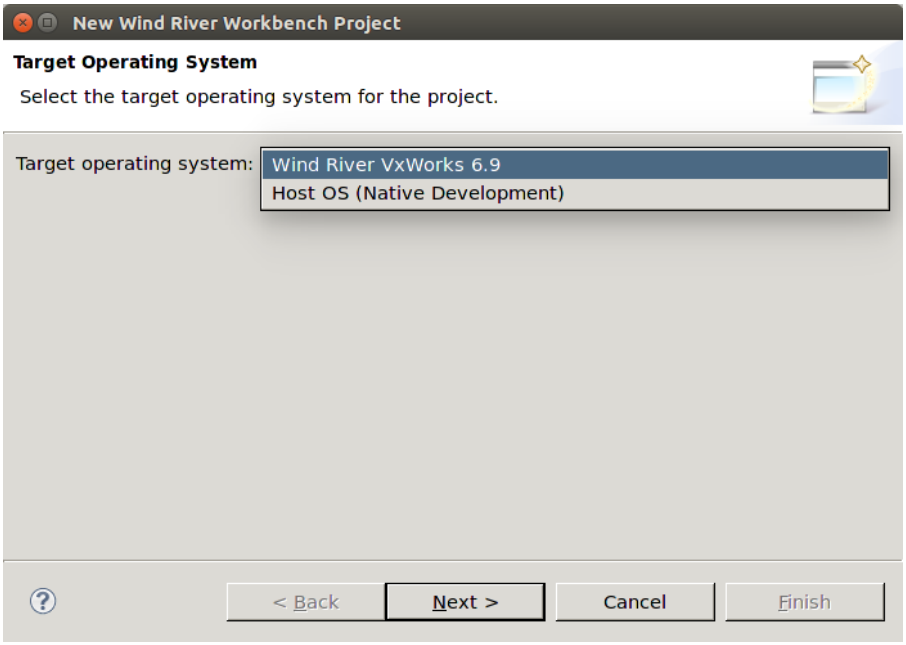
4.3.3.2 Using Workbench

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

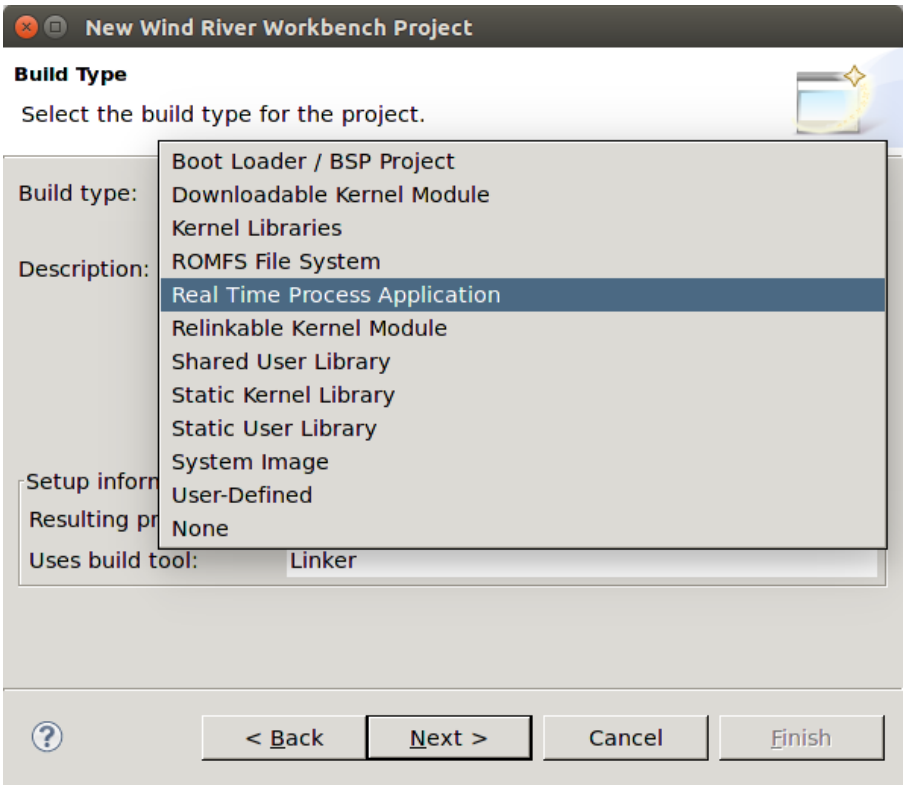
1. Start Workbench.
2. Select **File, New, Wind River Workbench Project**.



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



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



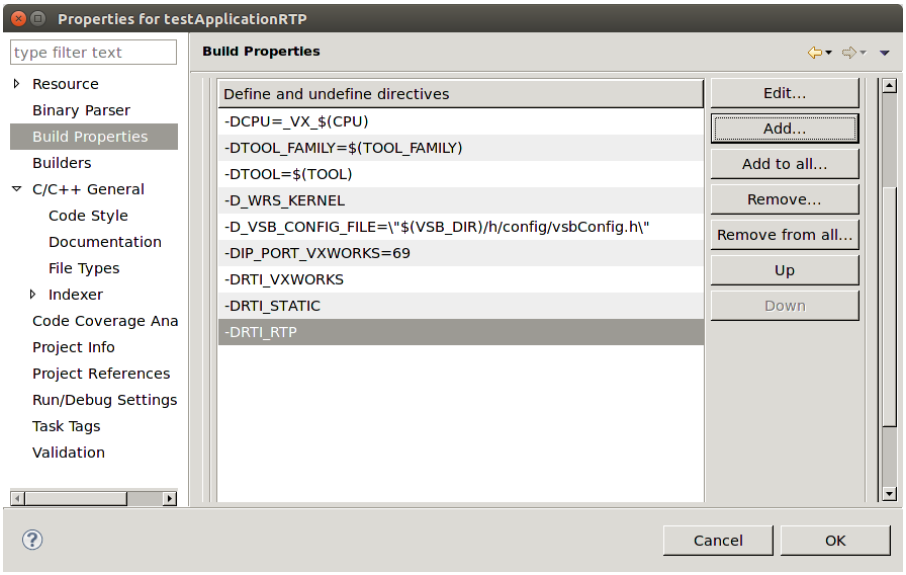
5. Give your project a name; click **Next**.
6. Leave everything else at its default setting; click **Finish**.

   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 17 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)*

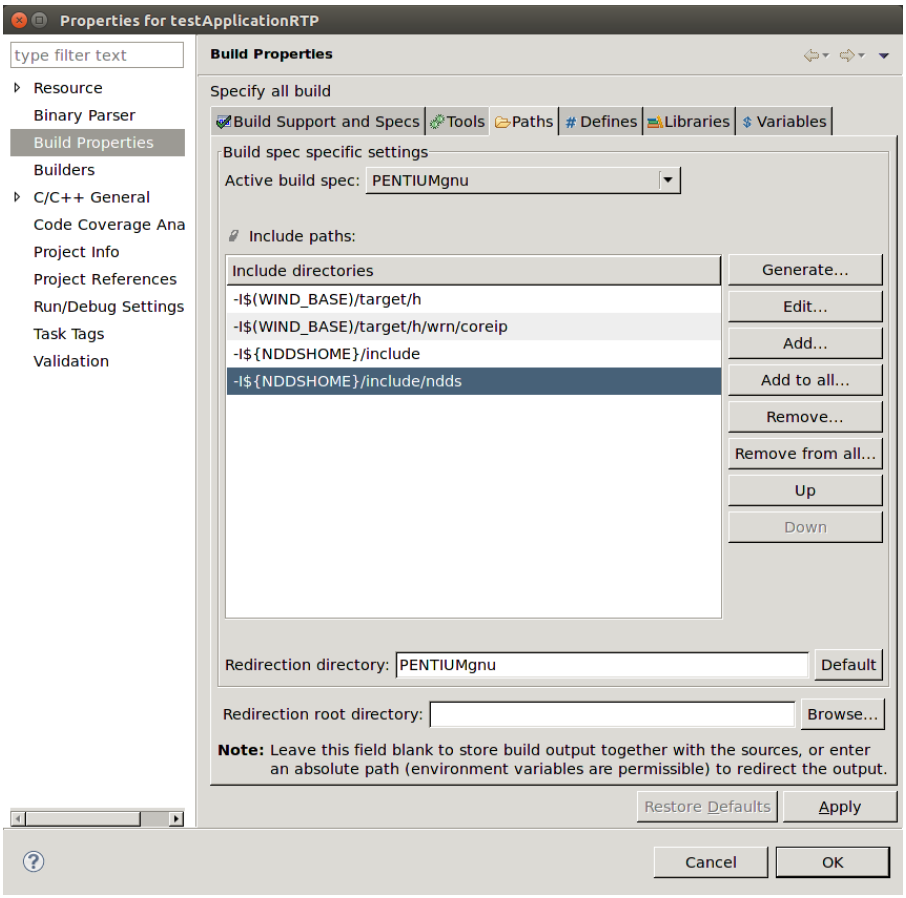
    Click **Apply** to save the changes.

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:

- The libraries **libnddscore.so**, **libnddsc.so**, and **libnddscpp.so** must first be loaded.
- 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).

In the examples below, 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, 0x8, 50000, rtiddsping, "-domainId 3 -publisher -
numSamples 100"
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-domainId 5 -subscriber -
timeout 20"
vxworks prompt> taskSpawn "rtiddsping", 100, 0x8, 50000, rtiddsping, "-help"
```

Spy:

```
vxworks prompt> taskSpawn "rtiddsspy", 100, 0x8, 50000, rtiddsspy, "-domainId 3 -topicRegex
Alarm*"
vxworks prompt> taskSpawn "rtiddsspy", 100, 0x8, 50000, rtiddsspy, "-help"
```

Chapter 5 Getting Started on VxWorks 653 Platform v2.3 Systems

This section provides simple instructions on how to configure a kernel and run *Connext DDS* applications on a VxWorks 653 Platform v2.3 system. Please refer to the documentation provided by Wind River Systems for more information, as well as the VxWorks section of the *RTI Connext DDS Core Libraries Platform Notes*.

Developing a complete system typically involves the cooperation of developers who play the following principal roles:

- *A platform provider*, who develops the platform
- *An application developer*, who develops applications
- *A system integrator*, who designs and specifies the module, and integrates a set of applications with a platform to create a module

For more information on these roles, please see the *VxWorks 653 Configuration and Build Guide*.

This section assumes the above distribution of development responsibilities, with the *Connext DDS* Core Libraries being a part of the application. This section is targeted towards platform providers, application developers, and system integrators.

- **For platform providers**, this section indicates what your system must provide to *Connext DDS*. Platform providers must provide a platform that application developers will use to create the application. The provided platform must support worker tasks and the socket driver. For the actual list of components, refer to the *RTI Connext DDS Core Libraries Platform Notes*.
- **For application developers**, this section describes how to create *Connext DDS* applications. Application developers must use the platform provided by the platform provider. To

create a *Connext DDS* application, follow the steps to Generate example code with rtiddsgen. on page 45 through Configure properties for the application. on page 46

- **For system integrators**, this section describes how to combine the platform from the platform provider, and the application from the application developer, and create the system to be deployed. System integrators must create an integration project using the module OS and partition OS provided by the platform provider, and the application provided by the application provider. To create a system capable of running *Connext DDS* applications, the system integrator needs to create a ConfigRecord considering the requirements noted in 5.2 Creating Connext DDS Applications for VxWorks 653 v2.3 Platforms on the next page.
- **For someone creating a Connext DDS application**, this section provides an example from the ground up.

For tips on using RTI DDS Ping and Spy, see 4.4 Using DDS Ping and Spy on page 32.

5.1 Setting up Workbench for Building Applications

Follow the steps in one of the following sections, depending on which socket library you want to install:

5.1.1 Installing the Wind River Services Socket Library below

or

5.1.2 Installing the RTI Socket Library below

5.1.1 Installing the Wind River Services Socket Library

1. Install Workbench.
2. Install **partition_socket_driver_v1.3**. Follow instructions from Wind River for the installation.

   For this example, the following steps were used for the installation:
   a. Copy the socket driver files from Wind River to each BSP of interest. For example, for sbc8641Vx653-2.3gcc3.3.2, copy the socket driver files into **$(WIND_BASE)/target/config/wrSbc8641d**.
   b. Copy the socket library header files into **$(WIND_BASE)/target/vThreads/h** (no files should be replaced or overwritten).

5.1.2 Installing the RTI Socket Library

1. Install Workbench.
2. Install **vx_653_socket.<Connext DDS version>**.
   a. Copy the socket driver files from RTI to each BSP of interest. Once you extract the RTI Socket Library zip file into your <NDDSHOME> installation directory, copy the contents of

vx_653_socket.<Connext DDS version>\bsp\src into **$(WIND_BASE)/target/config/**<*BSP*> (choose your BSP of interest. For instance, wrSbc8641d).

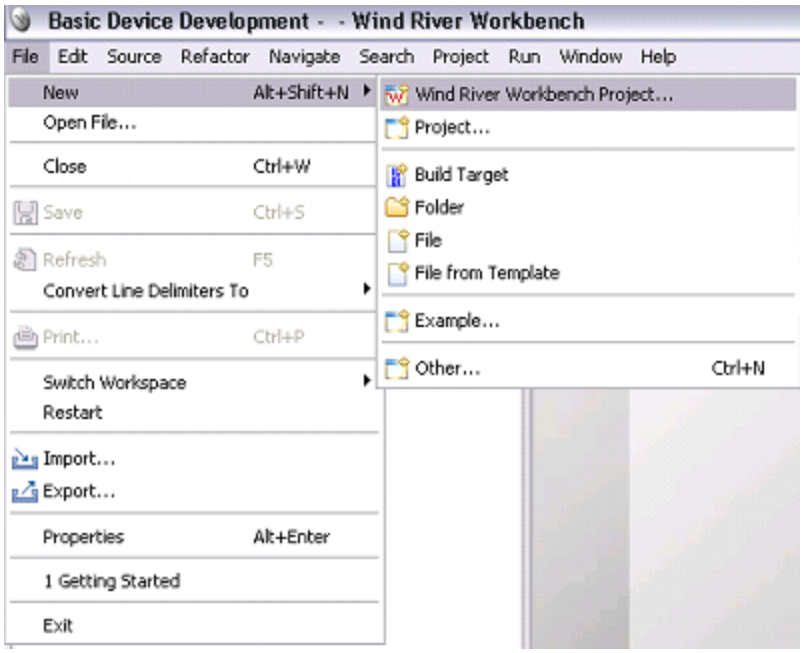
b. Link the **vx_653_socket.<Connext DDS version>** library to the application. You can find the libraries (release, debug, static, and dynamic) within your **NDDSHOME** installation directory. For example, for the dynamic release library, you would link **$NDDSHOME/-partition_os/lib/<architecture>/libvx_653_socket_posWrapper.so**.

5.2 Creating Connext DDS Applications for VxWorks 653 v2.3 Platforms

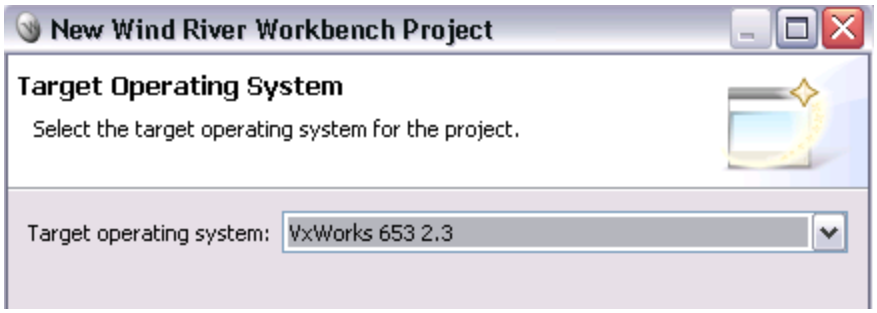
This section contains instructions for creating *Connext DDS* applications for the VxWorks 653 2.3 platforms (sbc8641Vx653-2.3gcc3.3.2 and simpcVx653-2.3gcc3.3.2). The screenshots show the process for sbc8641Vx653-2.3gcc3.3.2.

1. Create an integration project with two partitions (one for the publisher, one for the subscriber). Follow the instructions from Wind River for doing this. The following screenshots will guide you through the process.

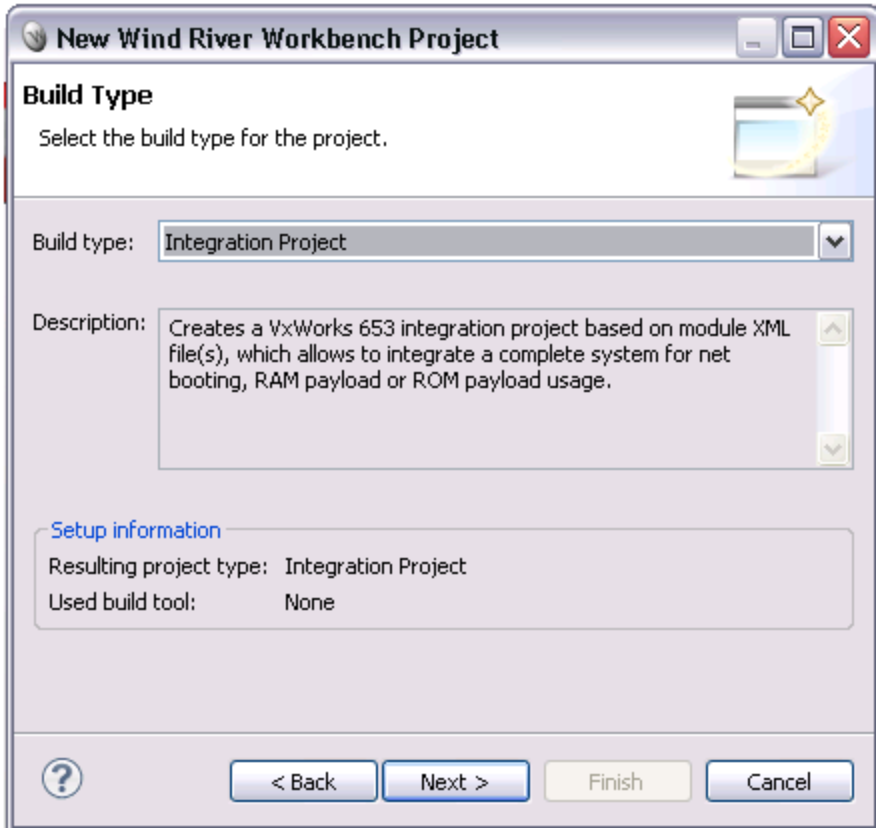
   a. Create a new Workbench project.



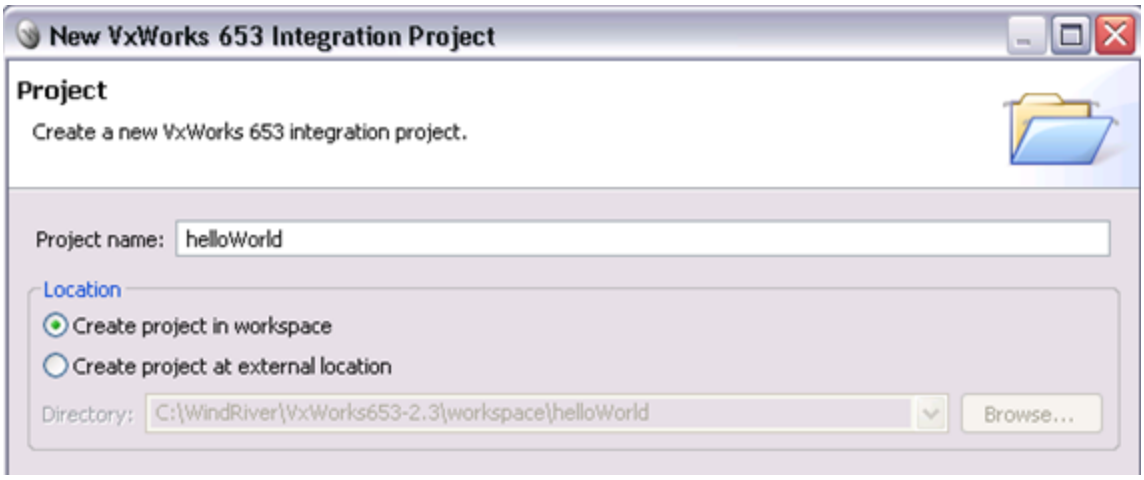
   b. For the Target operating system, select **VxWorks 653 2.3**.



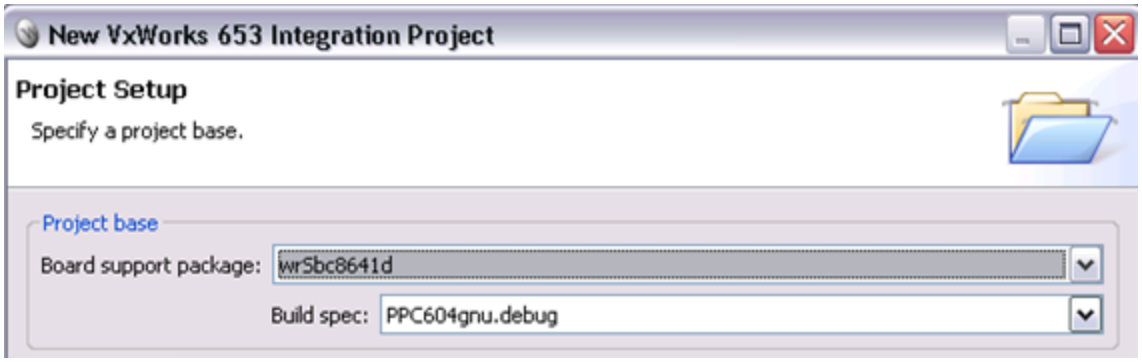
c. For Build type, select **Integration Project**.



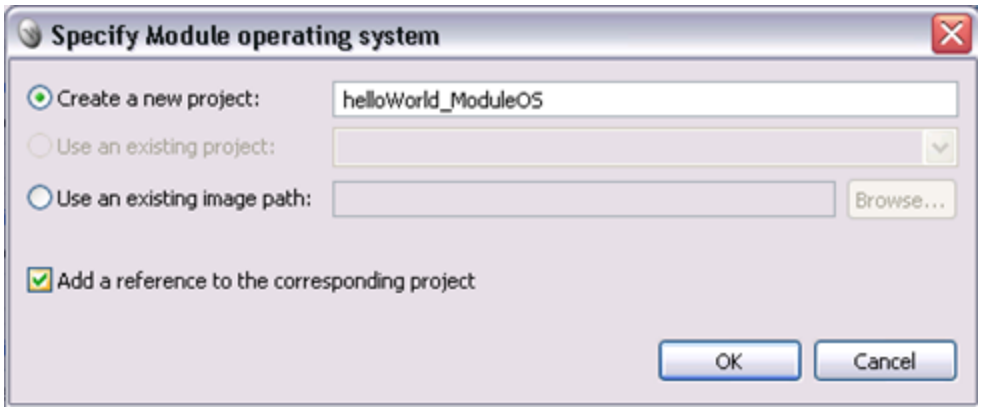
d. Create a project named **helloWorld** in the workspace.



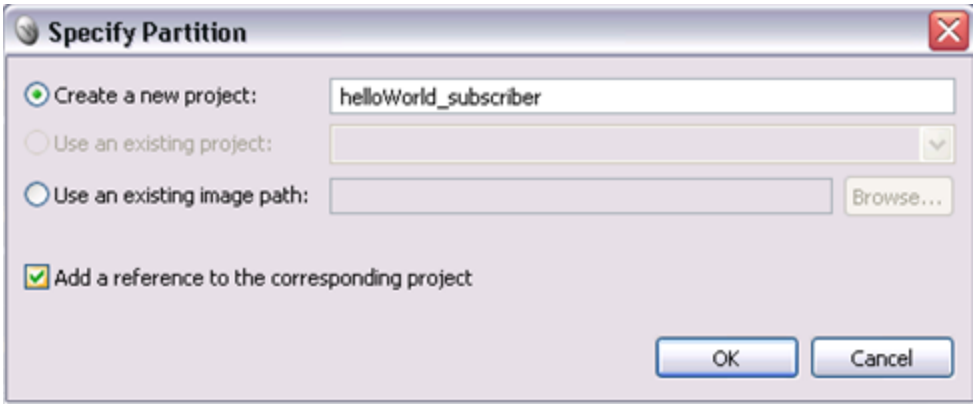
e. Select the appropriate Board Support package. Make sure the debug Build spec is selected. This example assumes the **wrSbc8641d** board support package is selected; alternatively, you could select **simpc**.



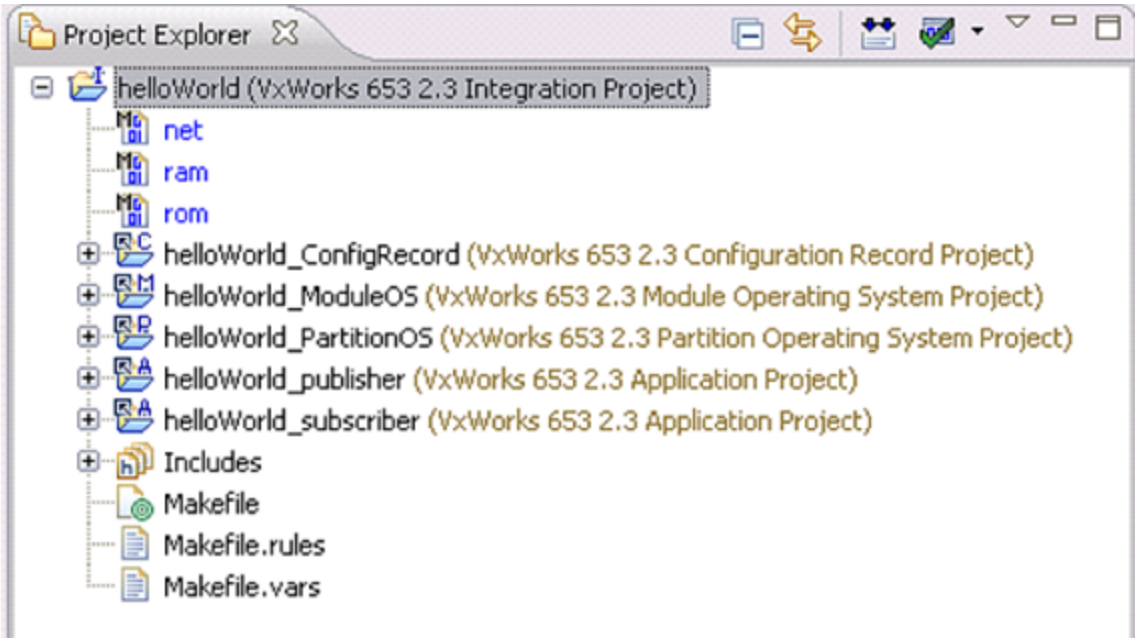
f. Select the default options for adding the ConfigRecord, ModuleOS, and PartitionOS. Make sure the "**Add a reference to the corresponding project**" checkbox is selected.



g. Create two partitions, **helloWorld_publisher** and **helloWorld_subscriber**, to create a Publisher and a Subscriber application, respectively. Make sure the **"Add a reference to the corresponding project"** checkbox is selected.

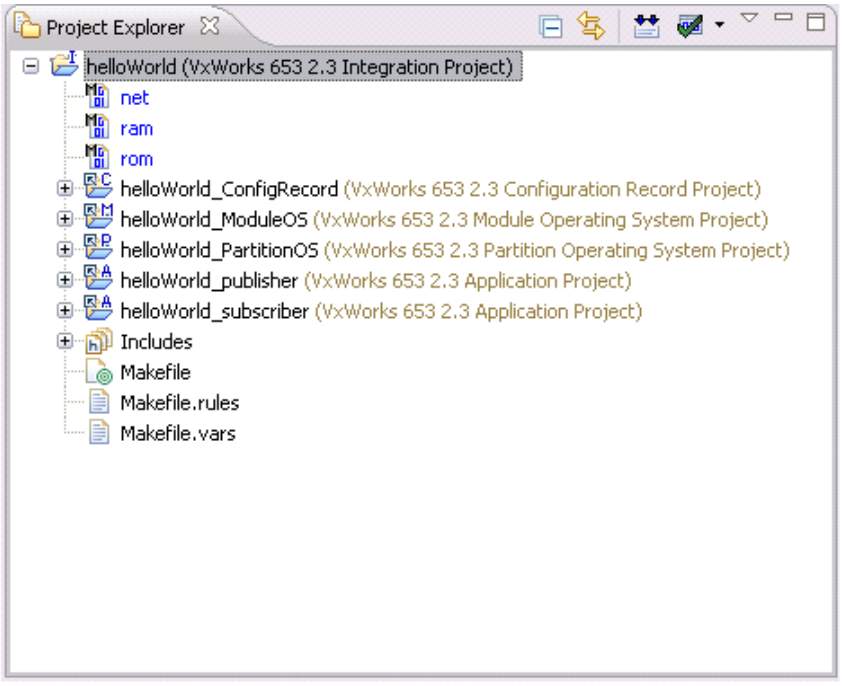


h. Now you are ready to create the Integration Project.



i. Click **Finish** to create the Integration project.

This will create an integration project with **ConfigRecord**, **ModuleOS**, **PartitionOS** and two partitions, **helloWorld_publisher** and **helloWorld_subscriber**.



2. Depending on your platform, open either **helloWorld_ConfigRecord/wrSbc8641d_default.xml** or **simpc_default.xml** and make the changes noted below. By default, the file opens in design mode.

You may wish to switch to source mode, which makes it easier to copy and paste sections, which is required in later steps.

a. Under Applications:

- Change the application name from **wrSbc8641d_part1** or **simpc_part1** to **helloWorld_publisher**.

  Note: Your application name should not be greater than 30 characters.

- In **MemorySize**, make these changes, depending on your platform:

	sbc8641Vx653-2.3gcc3.3.2	simpcVx653-2.3gcc3.3.2
MemorySizeBSS	0x5000	No change (keep default of 0x10000)
MemorySizeText	0x7F0000	0x640000
MemorySizeData	0x2000	No change (keep default of 0x10000)
MemorySizeRoData	0xE0000	0xf0000

For C++ only:

Change the **MemorySize** tag so it ends with '>' (not '/>').

For sbc8641Vx653-2.3gcc3.3.2: Within **MemorySize**, add:

```
<AdditionalSection Name=".gcc_except_table" Size="0x2000" Type="DATA"/>
```

For simpcVx653-2.3gcc3.3.2: Within **MemorySize**, add:

```
<AdditionalSection Name=".gcc_except_table" Size="0x10000" Type="DATA"/>
```

Remove **MemorySizePersistentData** and **MemorySizePersistentBss**.

Close **MemorySize** with **</MemorySize>**.

It should look like this when you are done:

For sbc8641Vx653-2.3gcc3.3.2:

```
<MemorySize MemorySizeBss="0x5000"
 MemorySizeText="0x7F0000"
 MemorySizeData="0x2000"
 MemorySizeRoData="0xE0000">
<AdditionalSection Name=".gcc_except_table"
 Size="0x2000" Type="DATA"/>
</MemorySize>
```

For simpcVx653-2.3gcc3.3.2:

```
<MemorySize MemorySizeBss="0x10000"
 MemorySizeText="0x640000"
 MemorySizeData="0x10000"
 MemorySizeRoData="0xf0000">
<AdditionalSection Name=".gcc_except_table"
 Size="0x10000" Type="DATA"/>
</MemorySize>
```

- Create a copy of the application **helloWorld_publisher** and rename it **helloWorld_subscriber**.

b. Under Partitions:

- Change the partition name from **wrSbc8641d_part1** or **simpc_part1** to **helloWorld_publisher**.
- Change the **Application NameRef** from **wrSbc8641d_part1** or **simpc_part1** to **helloWorld_publisher**.
- Under Settings, make these changes, depending on your platform:

	sbc8641Vx653-2.3gcc3.3.2	simpcVx653-2.3gcc3.3.2
RequiredMemorySize	0x2000000	0x2000000
numWorkerTasks	10	10

Create a copy of the partition application **helloWorld_publisher** and rename it **helloWorld_subscriber**. Change its **ID** to **2** and its **Application NameRef** to **helloWorld_subscriber**.

c. Under Schedules:

- Rename **PartitionWindow PartitionNameRef** from **wrSbc8641d_part1** or **simpc_part1** to **helloWorld_publisher**.
- Create a copy of the **PartitionWindow**, and change **PartitionNameRef** to **helloWorld_subscriber**.
- Add another **PartitionWindow**, with **PartitionNameRef** "**SPARE**" and Duration **0.05**. This partition window schedules the kernel, allowing time in the schedule for system activities like network communications.
- Optionally:
  i. If you want only *one* of the applications to run (**helloWorld_publisher** *or* **helloWorld_subscriber**), then you only need a partition window for the one you want to run.
  ii. If you do not want the *Connext DDS* application to run immediately when the system boots up, change the schedule ID to non-zero and add a SPARE schedule with ID 0.

d. Under HealthMonitor:

- In **PartitionHMTable Settings**, change **TrustedPartition NameRef** from **wrSbc8641d_part1** or **simpc_part1** to **helloWorld_publisher**. This is an optional field, so it can even be removed from the configuration.
- Optionally, change the **ErrorActions** from **hmDefaultHandler** to **hmDbgDefaultHandler**, in case you want the partitions to stop and not restart on exceptions.

e. Under Payloads:

- Change **PartitionPayload NameRef** from **wrSbc8641d_part1** or **simpc_part1** to **helloWorld_publisher**.
- Create a copy of the **PartitionPayload**, and change **NameRef** to **helloWorld_subscriber**.

f. Save the changes to **wrSbc8641d_default.xml** or **simpc_default.xml**, depending on your platform.

3. For simpcVx653-2.3gcc3.3.2 only:

a. Open **helloWorld_ConfigRecord/simpc.xml**.

b. Change the **PhysicalMemory Size** to 0x04000000.

c. In the **ramPayloadRegion** tag, change **Base_Address** to 0x23000000.

d. Change the **payloadMemory** Size to 0x2000000.

e. Save the changes to **simpc.xml**. After the changes, it should look like this:

```
<PhysicalMemory Size="0x04000000" Base_Address="0x20000000">
    <kernelMemoryRegion Size="0x00600000"/>
    <kernelConfigRecordRegion Size="0x00010000"/>
    <kernelPgPool Size="0x00200000"/>
    <portRegion Size="0x00200000"/>
    <hmLogRegion Size="0x00100000"/>
    <ramPayloadRegion Size="0x00000000" Base_Address="0x23000000"/>
    <aceMemoryRegion Size="0x00000000" Base_Address="0x20C00000"/>
    <userMemoryRegion Size="0x0b000000" Base_Address="0x20C00000"/>
</PhysicalMemory>
<payloadMemory Size="0x2000000" Base_Address="0x0"/>
```

4. Under **helloWorld_ModuleOS, Kernel Configuration**:

   a. Include the socket library component. Choose one of the following:

   - Include the Wind River Socket Library from
     hardware->peripherals->
     BSP configuration variants->
     Socket I/O Device [INCLUDE_SOCKET_DEV].

   Or

   - Include the RTI Socket Library from
     hardware->peripherals->
     BSP configuration variants->
     RTI's Socket I/O Device [INCLUDE_RTI_SOCKET_DEV].

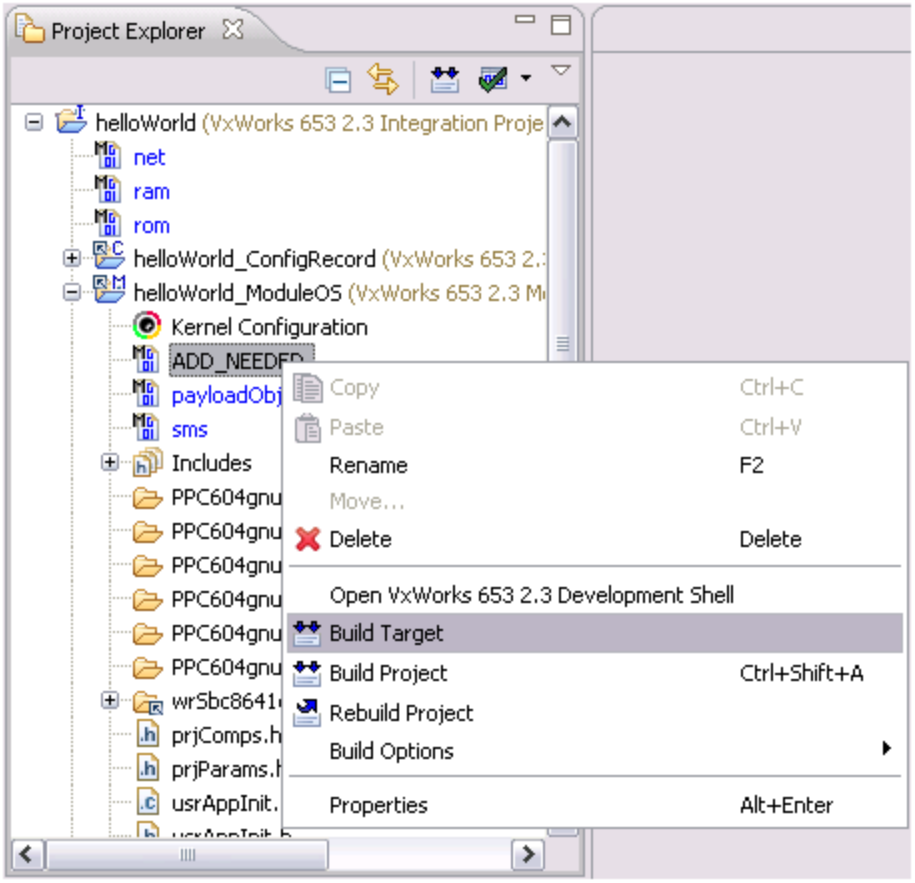
   b. Include **development tool components->**
   debug utilities [INCLUDE_DEBUG_UTIL]. This is needed to enable worker tasks.

   c. Optionally, include target-resident shell components, and any other components you want to include in the ModuleOS. Note that the target-resident shell component may be too large to include in SimPC without additional memory tuning.

   d. Save the changes to Kernel Configuration.

> See the *RTI Connext DDS Core Libraries Platform Notes* for a complete list of required kernel components for each platform.

5. Build the target **helloWorld_ModuleOS->ADD_NEEDED**.



6. Generate example code with *rtiddsgen*.

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

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

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

   c. Use *rtiddsgen* to generate sample code and a makefile, as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries 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
```

The supported values for <*architecture*> are listed in the *Release Notes* (**RTI_ConnextDDS_CoreLibraries_ReleaseNotes.pdf**), such as **sbc8641Vx653-2.3gcc3.3.2** or **simpcVx653-2.3gcc3.3.2**.

d. Edit the generated example code as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries Getting Started Guide.

7. Import the generated code into the application.

   a. Right-click **helloWorld_publisher** and select **Import**.

   b. In the Import wizard, select **General, File System**, then click **Next**.

   c. Browse to the **myhello** directory.

   d. Select the generated files, except **HelloWorld_subscriber**.

   e. *If and only if you are using the Wind River socket library*: import **sockLib.c** from the socket library into the project.

   f. Right-click **usrAppInit.c** and delete it.

   g. Repeat the same process for **helloWorld_subscriber**, this time importing **HelloWorld_subscriber** instead of **HelloWorld_publisher**.

8. Configure properties for the application.

   a. Right-click **helloWorld_publisher** and select **Properties**.

      i. Select **Build Properties** in the selection list on the left.

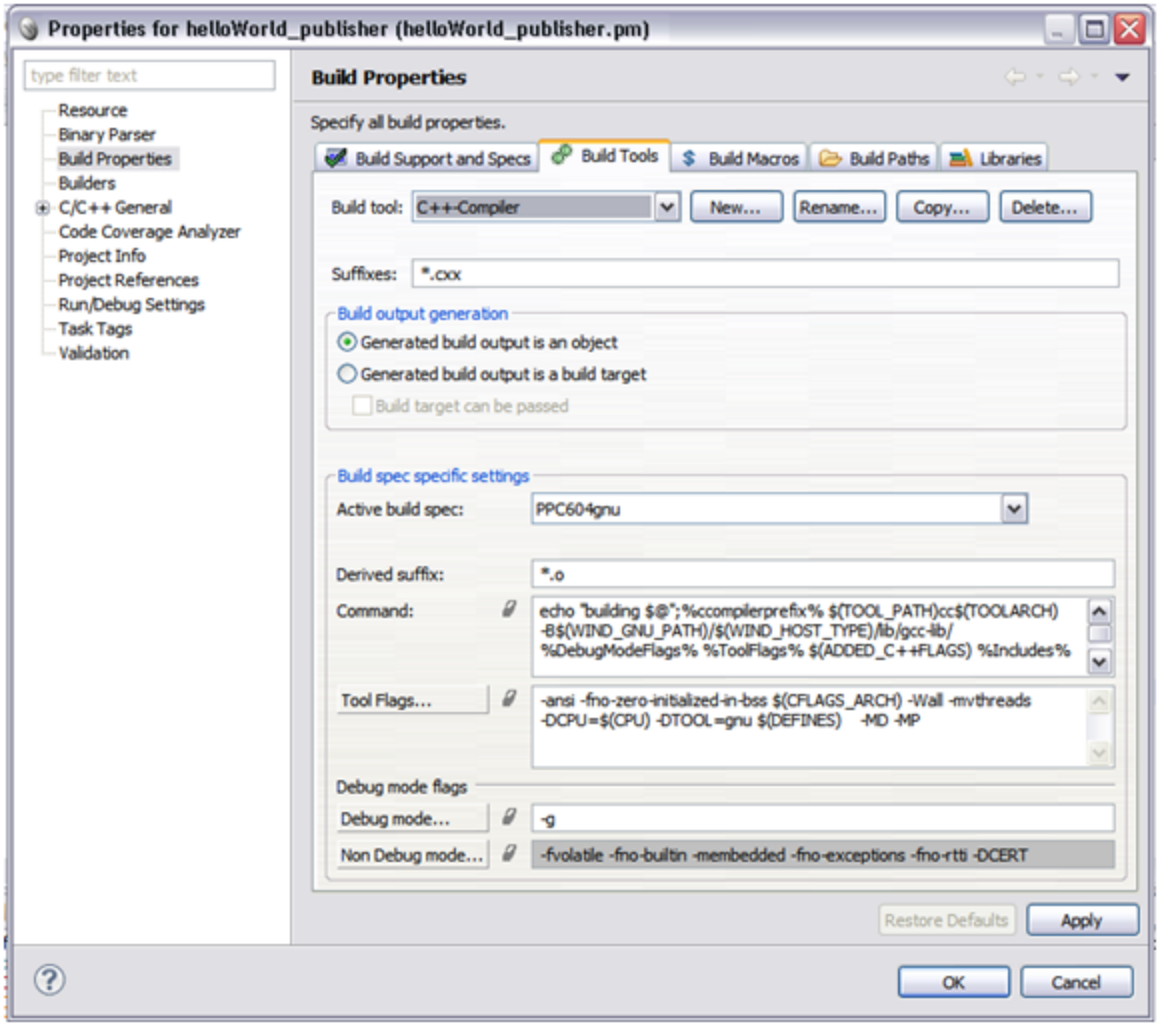
      ii. In the Build Macros tab:

      - Add a new macro, **NDDSHOME**, and set its value to the location where *Connext DDS* is installed. If this is in a directory with spaces in the path (such as Program Files), put quotation marks around the whole path. For the path, use forward slashes ("/"), not backslashes ("\").
      - Change the BLACKBOX value to **helloWorld_publisher**.

      iii. For C++ only:

      - In the Build Tools tab, select Build tool: **C++-Compiler**.
      - Change Suffixes to ***.cxx**.

      iv. Click **OK**.

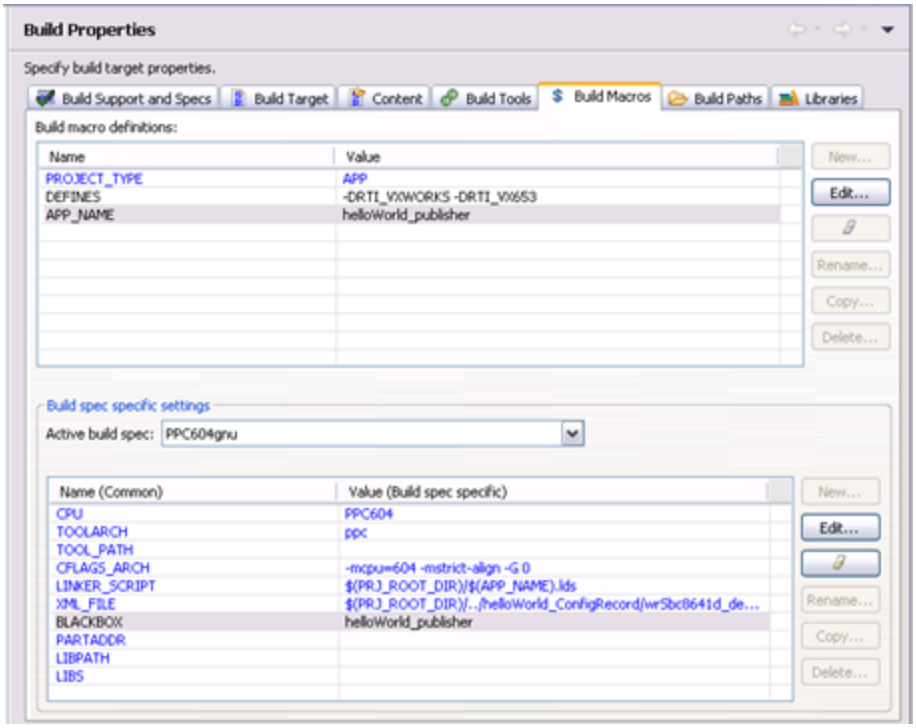


b. For C: Right-click **helloWorld_publisher**.

   For C++: Right-click **helloWorld_publisher, Build Targets, helloWorld_publisher-pm**.

c. Select **Properties**.

d. In the Build Macros tab, add **-DRTI_VXWORKS -DRTI_VX653** to DEFINES.



e. In the Build Paths tab, select the appropriate 'Active Build Spec' setting (such as PPC604gnu or SIMNTgnu). Then add these include directories, depending on your platform:

- **sbc8641Vx653-2.3gcc3.3.2:**

  -I$(WIND_BASE)/target/config/wrSbc8641d

  -I$(NDDSHOME)/include

  -I$(NDDSHOME)/include/ndds

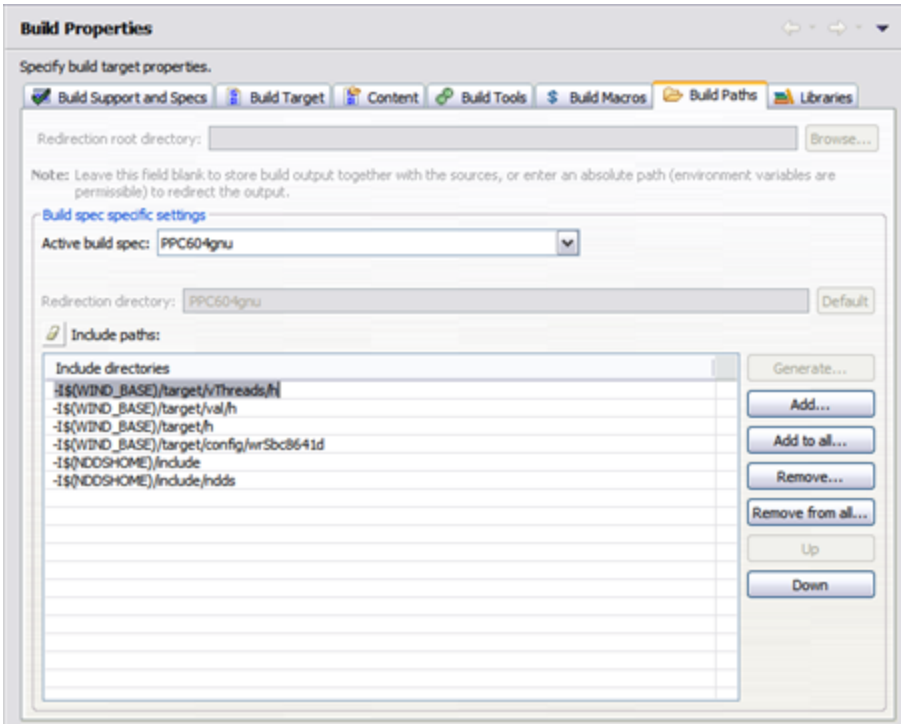
- **simpcVx653-2.3gcc3.3.2**

  -I$(WIND_BASE)/target/config/simpc

  -I$(NDDSHOME)/include

  -I$(NDDSHOME)/include/ndds

For **sbc8641Vx653-2.3gcc3.3.2**, the Build Paths tab will look like this:



f. In the Libraries tab:

Add the following files, depending on your platform and language:

sbc8641Vx653-2.3gcc3.3.2	simpcVx653-2.3gcc3.3.2
For C++ Only: $(WIND_BASE)/target/vThreads/lib/objPPC604gnuvx/vThreadsCplusComponent.o	For C++ Only: $(WIND_BASE)/target/vThreads/lib/objSIMNTgnuvx/vThreadsCplusComponent.o
For C++ Only: $(WIND_BASE)/target/vThreads/lib/objPPC604gnuvx/vThreadsCplusLibraryComponent.o	For C++ Only: $(WIND_BASE)/target/vThreads/lib/objSIMNTgnuvx/vThreadsCplusLibraryComponent.o
For all languages: $(NDDSHOME)/lib/sbc8641Vx653-2.3gcc3.3.2/libnddscore.so $(NDDSHOME)/lib/sbc8641Vx653-2.3gcc3.3.2/libnddsc.so	For all languages: $(NDDSHOME)/lib/simpcVx653-2.3gcc3.3.2/libnddscore.so $(NDDSHOME)/lib/simpcVx653-2.3gcc3.3.2/libnddsc.so
For C++ Only: $(NDDSHOME)/lib/sbc8641Vx653-2.3gcc3.3.2/libnddscpp.so	For C++ Only: $(NDDSHOME)/lib/simpcVx653-2.3gcc3.3.2/libnddscpp.so

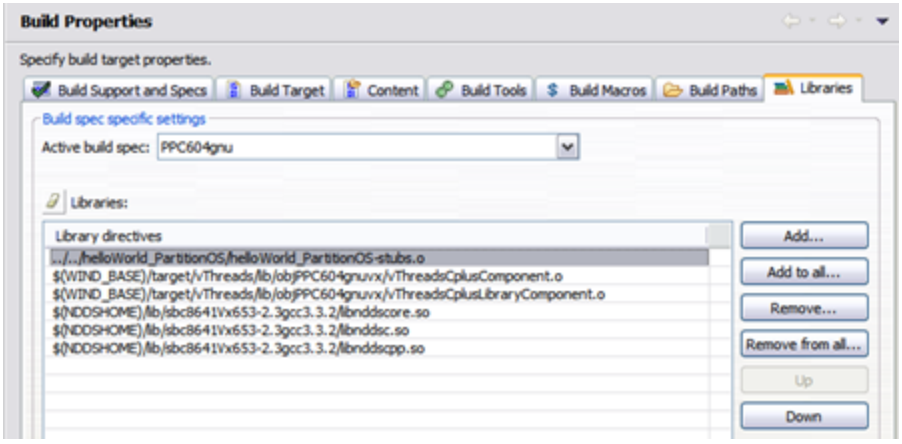
Make sure you have added the libraries as fully qualified names (without **-l** or **-L**).

If and only if you are using the RTI socket library: Add one of the following libraries to link with. This is an example for sbc8641Vx653-2.3gcc3.3.2:

Dynamic release	$(NDDSHOME)/partition_os/lib/ sbc8641Vx653-2.3gcc3.3.2/ libvx_653_socket_posWrapper.so
Dynamic debug	$(NDDSHOME)/partition_os/lib/ sbc8641Vx653-2.3gcc3.3.2/ libvx_653_socket_posWrapperd.so
Static release	$(NDDSHOME)/partition_os/lib/ sbc8641Vx653-2.3gcc3.3.2/ libvx_653_socket_posWrapperz.a
Static debug	$(NDDSHOME)/partition_os/lib/ sbc8641Vx653-2.3gcc3.3.2/ libvx_653_socket_posWrapperzd.a

g. Click **OK**.

For sbc8641Vx653-2.3gcc3.3.2 and the Wind River socket library, it should look like this:



For sbc8641Vx653-2.3gcc3.3.2 and the RTI socket library, it should look like the above image plus the RTI socket library.

h. Repeat the same process for **helloWorld_subscriber**.

9. Build the Integration Project.

5.3 Running Connext DDS Applications on an Sbc8641d Target

1. Boot up your target board with the kernel created by the Integration project.
2. If the *Connext DDS* applications are in schedule 0, they will start up automatically, and you should see the publisher and subscriber communicating with each other.

3. If the *Connext DDS* applications are not in schedule 0, use this command to change to the desired schedule: **arincSchedSet <Schedule number>**.

Chapter 6 Getting Started on VxWorks 653 v2.5.x Systems

This chapter provides simple instructions on how to configure a kernel and run *Connext DDS* applications on a VxWorks 653 version 2.5.x system. It shows specifically a VxWorks 653 2.5.0.1 example, which should also serve as a guide for other 2.5.x versions. Please refer to the documentation provided by Wind River Systems for more information, as well as the *RTI Core Libraries and Utilities Custom Support for VxWorks 653 Version 2.5 Platforms* (**RTI_ConnextDDS_CoreLibraries_PlatformNotes_VxWorks653_v2.5.pdf**).

Note: The memory settings in this document are specifically for the examples shown. Each version of *Connext DDS* will likely require updated memory settings. You will find these memory settings in the VxWorks section of the RTI Connext DDS Core Libraries Platform Notes for each version.

Developing a complete system typically involves the cooperation of developers who play the following principal roles:

- *A platform provider*, who develops the platform
- *An application developer*, who develops applications
- *A system integrator*, who designs and specifies the module, and integrates a set of applications with a platform to create a module

For more information on these roles, please see the *VxWorks 653 Configuration and Build Guide*.

This document assumes the above distribution of development responsibilities, with the *Connext DDS* Core Libraries being a part of the application. This document is targeted towards platform providers, application developers, and system integrators.

- **For platform providers**, this chapter indicates what your system must provide to *Connext DDS*. Platform providers must provide a platform that application developers will use to create the application. The provided platform must support worker tasks and the socket driver.

For the actual list of components, refer to Table 9.3, "*Building Instructions for VxWorks 653 Architectures*," in the *Platform Notes*.

- **For application developers**, this chapter describes how to create *Connext DDS* applications. Application developers must use the platform provided by the platform provider. To create a *Connext DDS* application, follow the steps in 6.1 Creating Connext DDS Applications for VxWorks 653 2.5.x below (start with the step to Generate example code with rtiddsgen. on page 61, through the step to Configure properties for the application. on page 62).
- **For system integrators**, this document describes how to combine the platform from the platform provider, and the application from the application developer, and create the system to be deployed. System integrators must create an integration project using the module OS and partition OS provided by the platform provider, and the application provided by the application provider. To create a system capable of running *Connext DDS* applications, the system integrator needs to create a ConfigRecord considering the requirements noted in the step to Edit the XML file in 6.1 Creating Connext DDS Applications for VxWorks 653 2.5.x below.
- **For someone creating a Connext DDS application**, this document provides an example from the ground up.

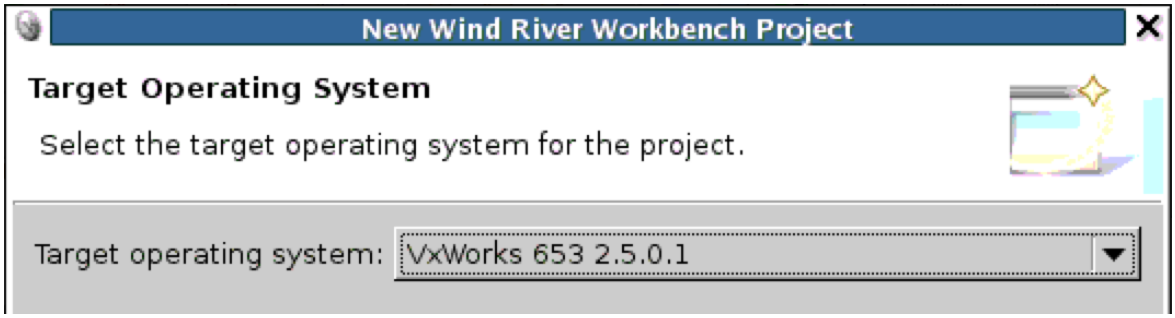
For tips on using RTI DDS Ping and Spy, see 4.4 Using DDS Ping and Spy on page 32.

6.1 Creating Connext DDS Applications for VxWorks 653 2.5.x

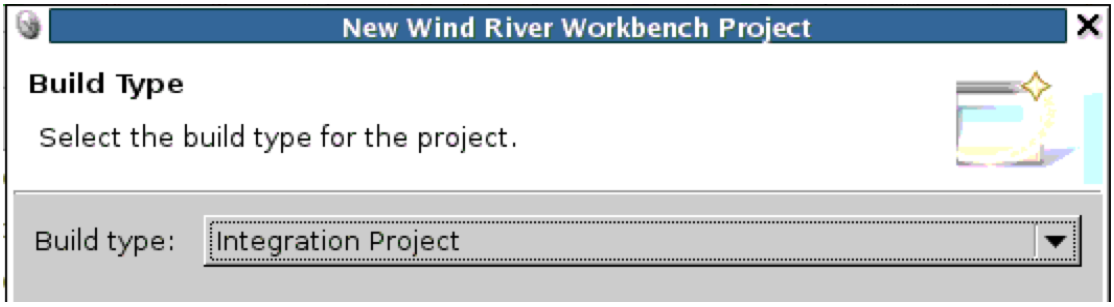
This section contains instructions for creating *Connext DDS* applications for the VxWorks 653 v2.5.0.1 platforms (ppce500v2Vx653-2.5gcc4.3.3). The screenshots show the process for this specific platform and version of VxWorks. Note that these instructions will vary from those for other VxWorks 653 versions, such as v2.3 and others.

1. Create an integration project with two partitions (one for the publisher, one for the subscriber). Follow the instructions from Wind River for doing this. The following screenshots will guide you through the process.
    a. Create a new Workbench project.

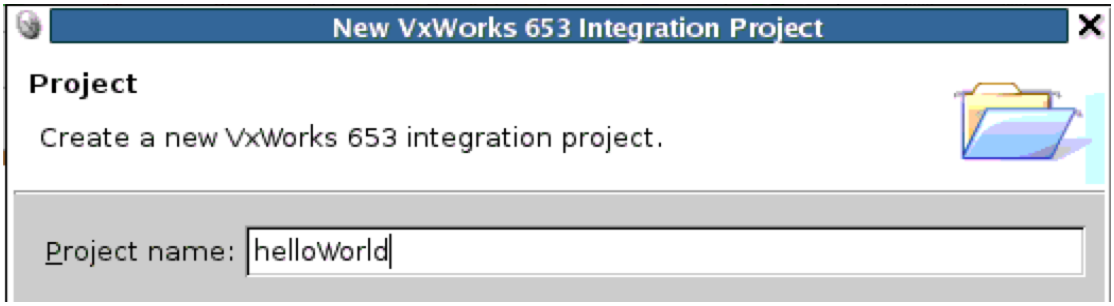
b. For the Target operating system, select **VxWorks 653 2.5.0.1**.



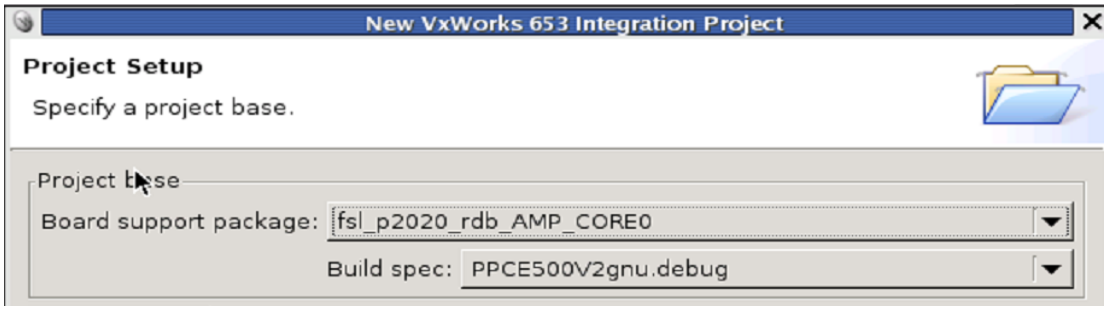
c. For Build type, select **Integration Project**.



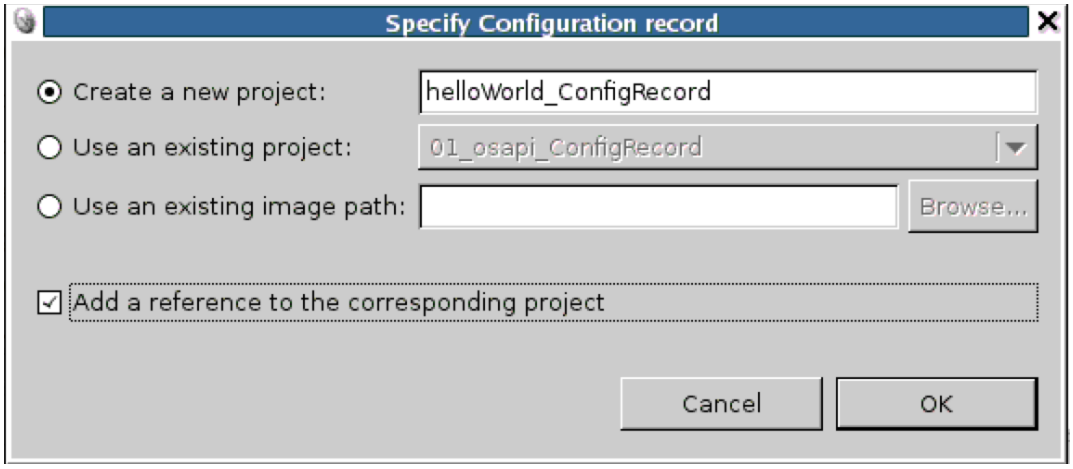
d. Create a project named **helloWorld** in the workspace.

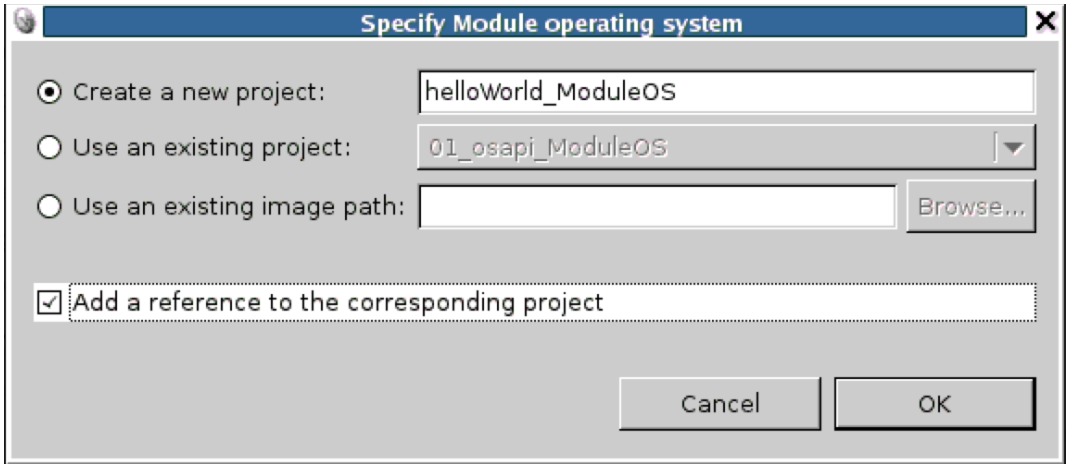


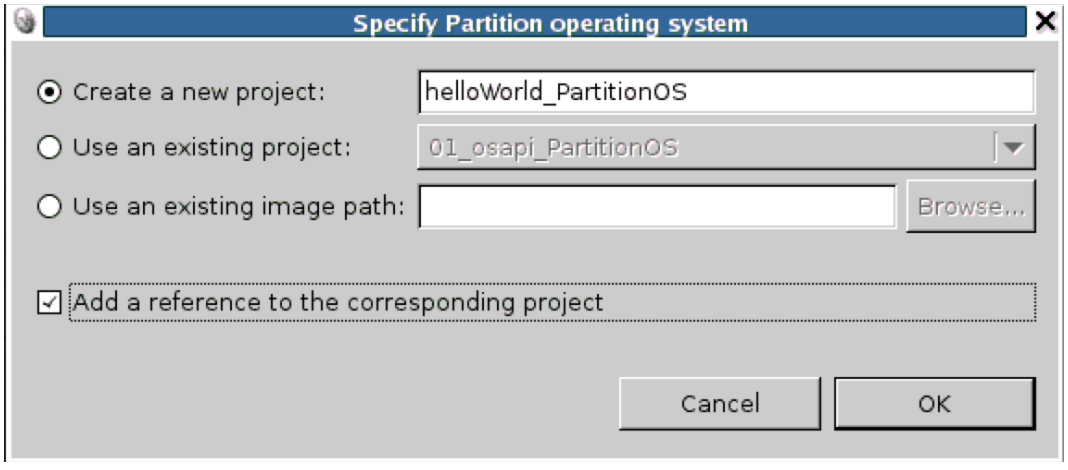
e. Select the appropriate Board Support package. Make sure the debug Build spec is selected. This example assumes the fsl_p2020_rdb_AMP_CORE0 board support package is selected.



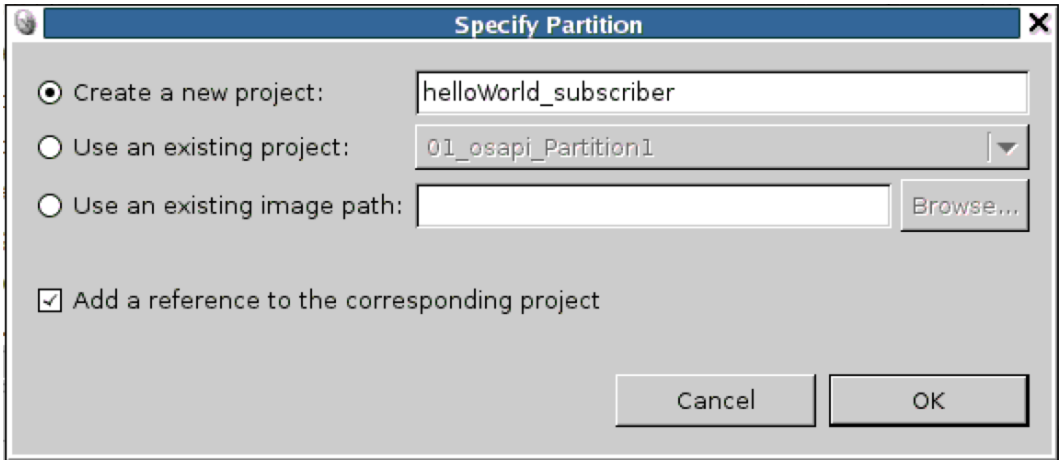
f. Select the default options for adding the ConfigRecord, ModuleOS, and PartitionOS. Make sure the "**Add a reference to the corresponding project**" check box is selected.



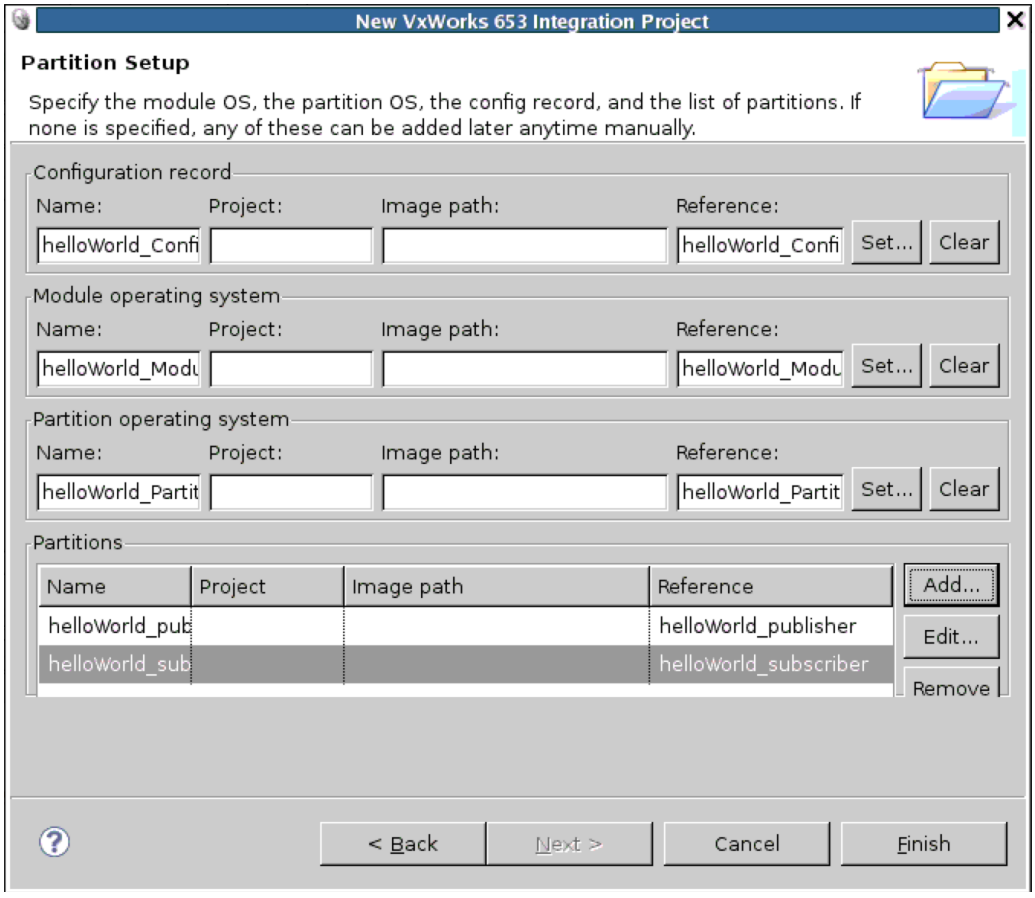




g. Create two partitions, **helloWorld_publisher** and **helloWorld_subscriber**, to create a Publisher and a Subscriber application, respectively. Make sure the **"Add a reference to the corresponding project"** check box is selected.

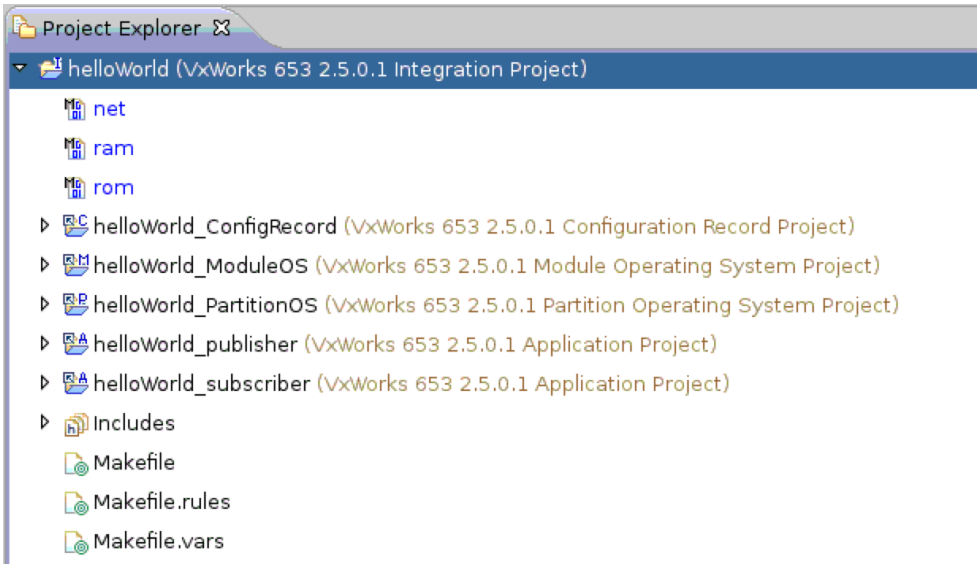


h. Now you are ready to create the Integration Project.

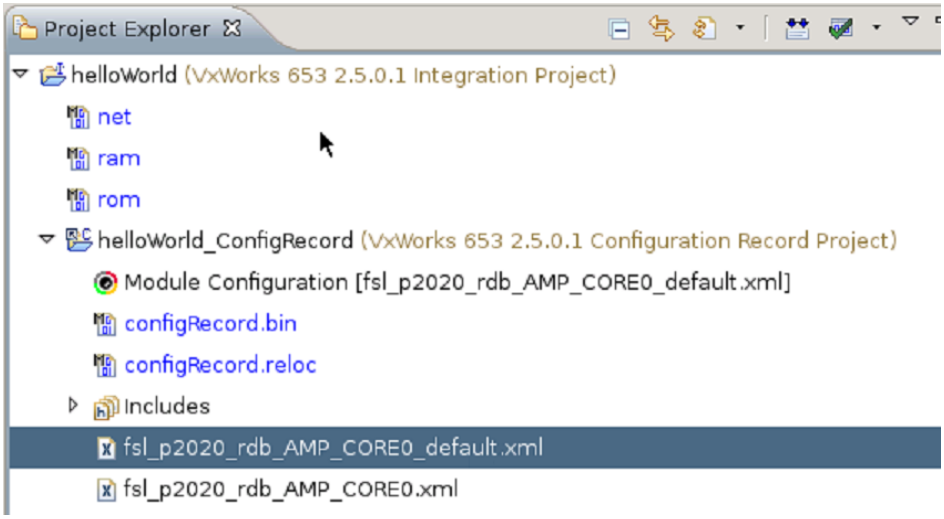


i. Click **Finish** to create the Integration project.

This will create an integration project with **ConfigRecord**, **ModuleOS**, **PartitionOS** and two partitions, **helloWorld_publisher** and **helloWorld_subscriber**.



2. Edit the XML file. Depending on your platform, open **fsl_b4860_qds_AMP_CORE0_default.xml** and make the changes noted below. By default, the file opens in design mode. You may want to switch to source mode, which makes it easier to copy and paste sections, which is required in later steps.



   a. Under Applications:
      i. Change the application name from **fsl_b4860_qds_AMP_CORE0_part1** to **helloWorld_publisher**.
      ii. In **MemorySize**, make these changes:

- MemorySizeBss = "0x5000"
- MemorySizeText = "0xBFF000"
- MemorySizeData = "0xf000"
- MemorySizeRoData = "0xff000"

It should look like this when you are done:

```
<MemorySize MemorySizeBss="0x5000"
               MemorySizeText="0xBFF000"
               MemorySizeData="0xf000"
               MemorySizeRoData="0xff000"/>
```

iii. Create a copy of the application **helloWorld_publisher** and rename it **helloWorld_subscriber**.

iv. Change the application name from **fsl_b4860_qds_AMP_CORE0_part1** to **helloWorld_publisher**.

v. In **MemorySize**, make these changes:

- MemorySizeBss = "0x5000"
- MemorySizeText = "0xBFF000"
- MemorySizeData = "0xf000"
- MemorySizeRoData = "0xff000"

It should look like this when you are done:

```
<MemorySize MemorySizeBss="0x5000"
               MemorySizeText="0xBFF000"
               MemorySizeData="0xf000"
               MemorySizeRoData="0xff000"/>
```

vi. Create a copy of the application **helloWorld_publisher** and rename it **helloWorld_subscriber**.

b. Under Shared LibraryRegions, change MemorySize MemorySizeBss to 0x6000.

c. Under Partitions:

i. Change the partition name from **fsl_b4860_qds_AMP_CORE0_part1** to **helloWorld_publisher**.

ii. Change the **Application NameRef** from **fsl_b4860_qds_AMP_CORE0_part2** to **helloWorld_publisher**.

iii. Under Settings, make these changes:

- RequiredMemorySize = "0x1000000"
- numWorkers = "10"
- maxGlobalFDs = "50"

iv. Create a copy of the partition application **helloWorld_publisher** and rename it **helloWorld_subscriber**. Change its **ID** to **2** and its **Application NameRef** to **helloWorld_subscriber**.

d. Under Schedules:

i. Rename **PartitionWindow PartitionNameRef** from **fsl_b4860_qds_AMP_CORE0_part1** to **helloWorld_publisher**.

ii. Create a copy of the **PartitionWindow** and change **PartitionNameRef** to **helloWorld_subscriber**.

iii. Add another **PartitionWindow**, with **PartitionNameRef** "**SPARE**" and Duration **0.05**. This partition window schedules the kernel, allowing time in the schedule for system activities like network communications.

iv. Optionally:

- If you want only *one* of the applications to run (**helloWorld_publisher** *or* **helloWorld_subscriber**), then you only need a partition window for the one you want to run.
- If you do not want the *Connext DDS* application to run immediately when the system boots up, change the schedule ID to non-zero and add a SPARE schedule with ID 0.

e. Under HealthMonitor:

i. In **PartitionHMTable Settings**, change **TrustedPartition NameRef** from **fsl_b4860_qds_AMP_CORE0_part1** to **helloWorld_publisher**. This is an optional field, so it can even be removed from the configuration.

ii. Optionally, change the **ErrorActions** from **hmDefaultHandler** to **hmDbgDefaultHandler**, in case you want the partitions to stop and not restart on exceptions.

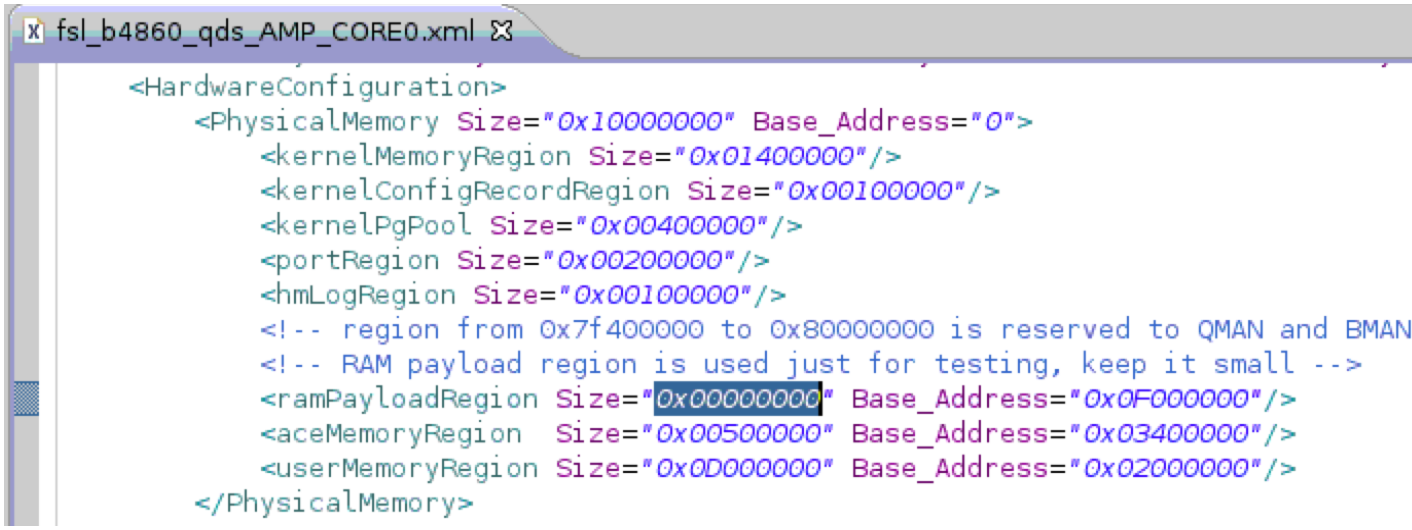
f. Under Payloads:

i. Change **PartitionPayload NameRef** from **fsl_b4860_qds_AMP_CORE0_part1** to **helloWorld_publisher**.

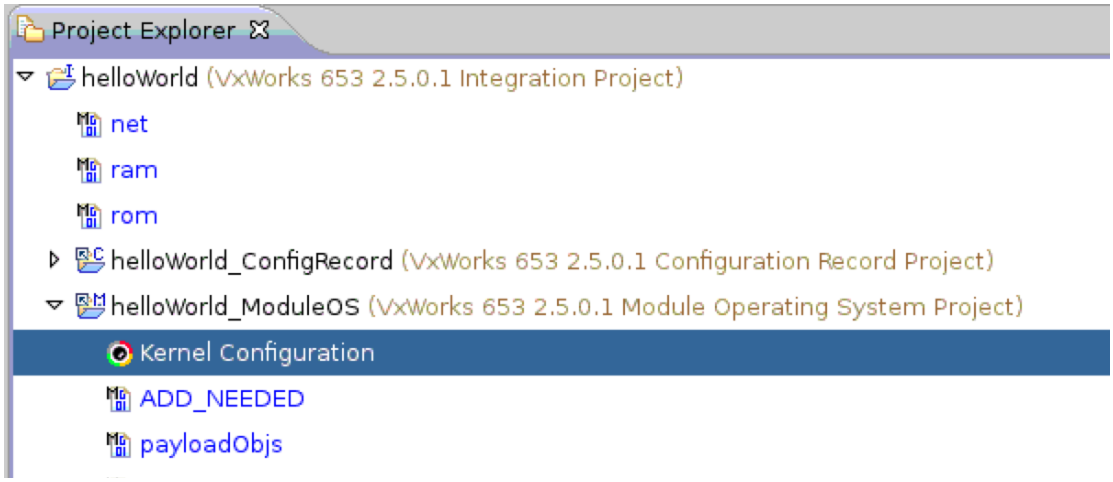
ii. Create a copy of the **PartitionPayload**, and change **NameRef** to **helloWorld_subscriber**.

e. Save the changes to **fsl_b4860_qds_AMP_CORE0_part1_default.xml**.

3. Depending on the project example you are using, you may need to set the **ramPayLoad** size to zero. If needed, go to the ConfigRecord project and modify the **<BSP>.xml** file (**fsl_p2020_rdb_AMP_CORE0.xml** in this example) and set the **rampPayloadRegion** size to zero. It should look like this after being modified:



4. Under **helloWorld_ModuleOS, Kernel Configuration**:



a. Include **network components->network private components->FACE POSIX support driver [INCLUDE_FACE_POSIX_SOCKET_DRV]**.

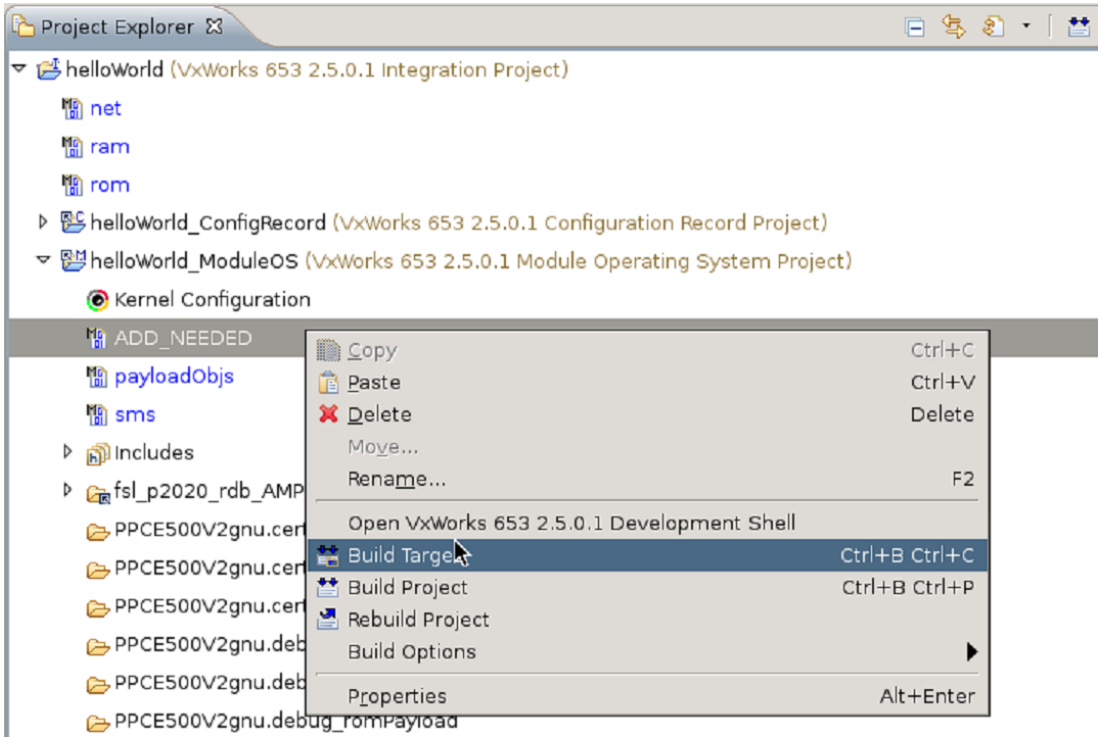
b. Include **development tool components->debug utilities [INCLUDE_DEBUG_UTIL]**. This is needed to enable worker tasks.

c. Optionally, include target-resident shell components and any other components you want to include in the ModuleOS. Note that the target-resident shell component may be too large and you may need additional memory tuning.

d. Save the changes to Kernel Configuration.

> See the *RTI Core Libraries Platform Notes* (**RTI_ConnextDDS_CoreLibraries_PlatformNotes.pdf**) for a complete list of required kernel components for each platform.

5. Build the target **helloWorld_ModuleOS->ADD_NEEDED**.



6. Generate example code with *rtiddsgen*.

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

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

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

c. Use *rtiddsgen* to generate sample code and a makefile, as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries Getting Started Guide. Choose either C or C++.

For C:

```
rtiddsgen -language C -example ppce500v2Vx653-2.5gcc4.3.3 HelloWorld.idl
```

For C++:

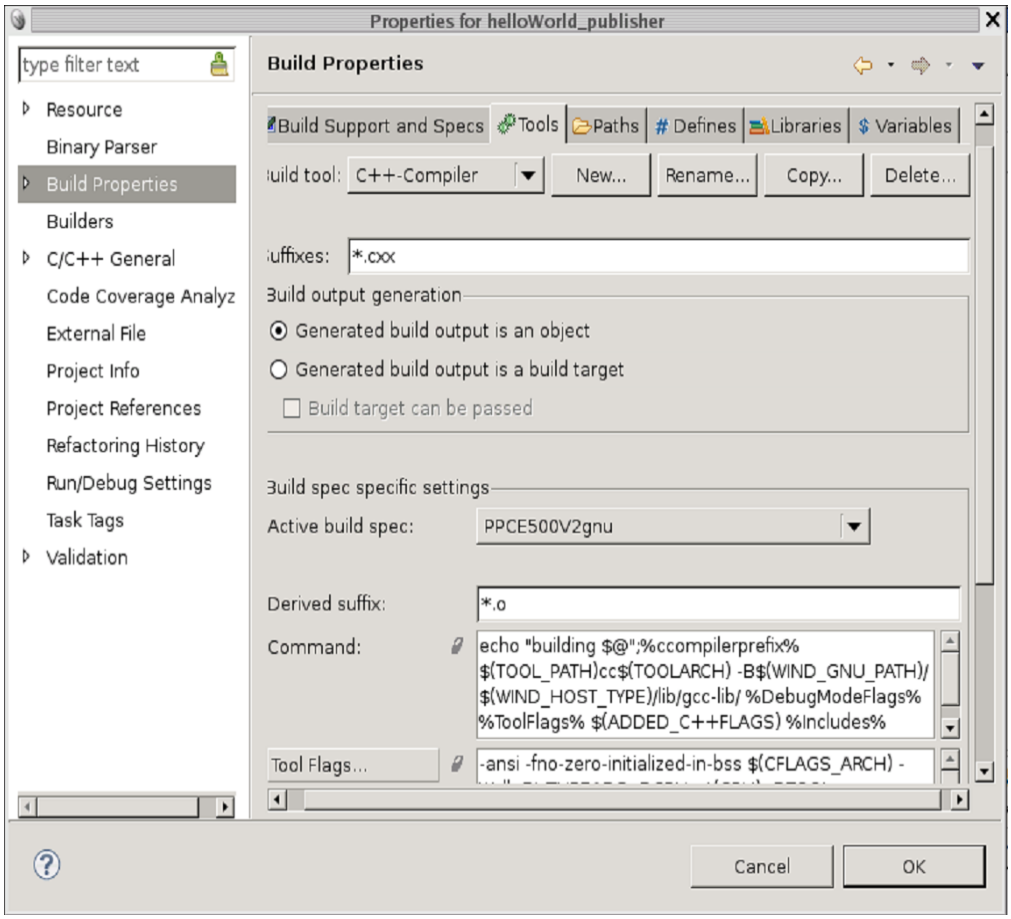
```
rtiddsgen -language C++ -example ppce500v2Vx653-2.5gcc4.3.3 HelloWorld.idl
```

For more information on the ppce500v2Vx653-2.5gcc4.3.3 architecture, please see the separate document, *RTI Core Libraries and Utilities Custom Support for VxWorks 653 Version 2.5 Platforms* (**RTI_ConnextDDS_CoreLibraries_PlatformNotes_VxWorks653_v2.5.pdf**).

d. Edit the generated example code as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries Getting Started Guide.

7. Import the generated code into the application.
   a. Right-click **helloWorld_publisher** and select **Import**.
   b. In the Import wizard, select **General, File System**, then click **Next**.
   c. Browse to the **myhello** directory.
   d. Select the generated files, except **HelloWorld_subscriber**.
   e. Right-click **usrAppInit.c** and delete it.
   f. Repeat the same process for **helloWorld_subscriber**, this time importing **HelloWorld_subscriber** instead of **HelloWorld_publisher**.
8. Configure properties for the application.
   a. Right-click **helloWorld_publisher** and select **Properties**.
      i. Select **Build Properties** in the selection list on the left.
      ii. In the Variables tab:
         - Add a new variable, **NDDSHOME**, and set its value to the location where *Connext DDS* is installed. If this is in a directory with spaces in the path (such as Program Files), put quotation marks around the whole path.
         - Change the BLACKBOX value to **helloWorld_publisher**.
      iii. For C++ only:
         - In the Tools tab, select Build tool: **C++-Compiler**.
         - Change Suffixes to ***.cxx**.

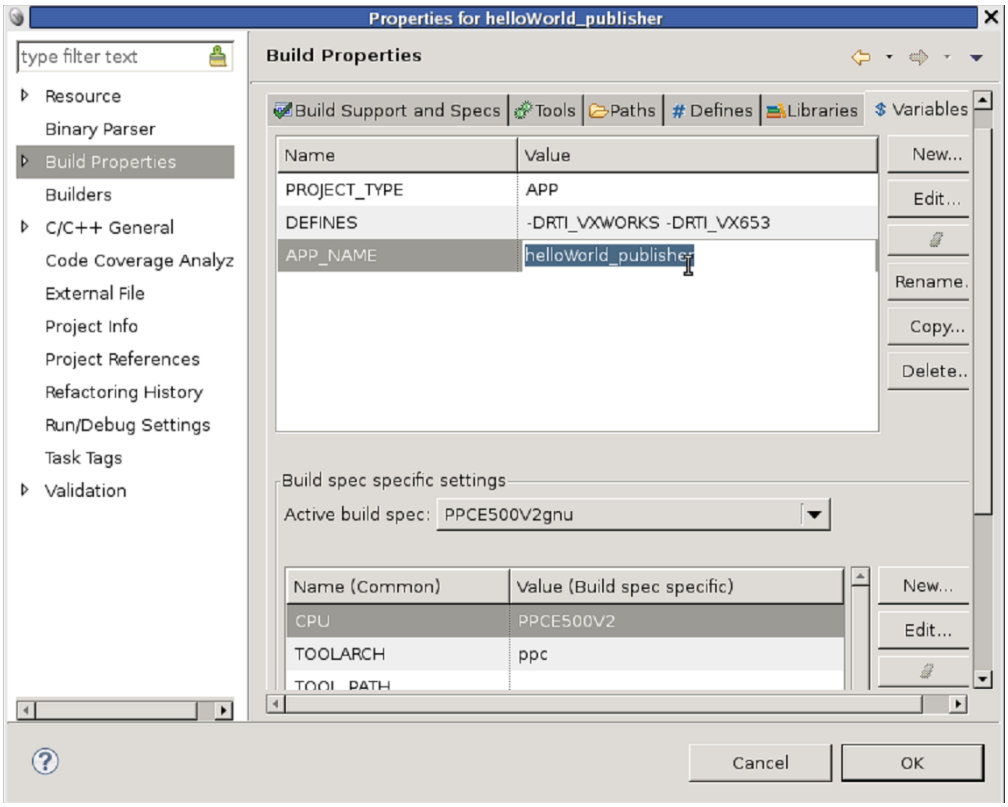
iv. Click **OK**.



b. For C: Right-click **helloWorld_publisher**.
   For C++: Right-click **helloWorld_publisher, Build Targets, helloWorld_publisher-pm**.

c. Select **Properties**.

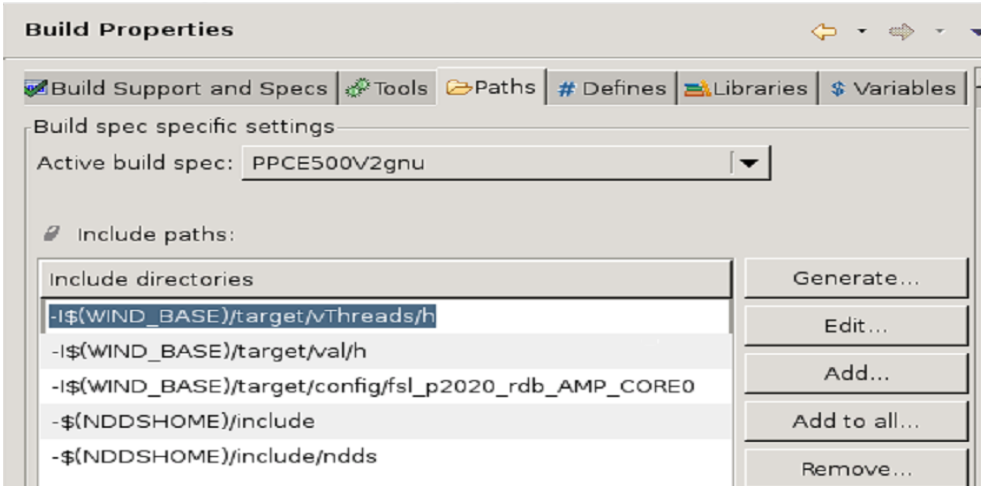
d. In the Variables tab, add -DRTI_VXWORKS -DRTI_VX653 to DEFINES.



e. In the Paths tab, select the appropriate 'Active Build Spec' setting (such as PPCE6500gnu). Then add these include directories:

- **-I$(WIND_BASE)/target/config/fsl_p2020_rdb_AMP_CORE0**
- **-I$(NDDSHOME)/include**
- **-I$(NDDSHOME)/include/ndds**

The Build Paths tab will look like this:



f. In the Libraries tab, add the following files, depending on your language. Note that in this example we use RTI's *dynamic* libraries:

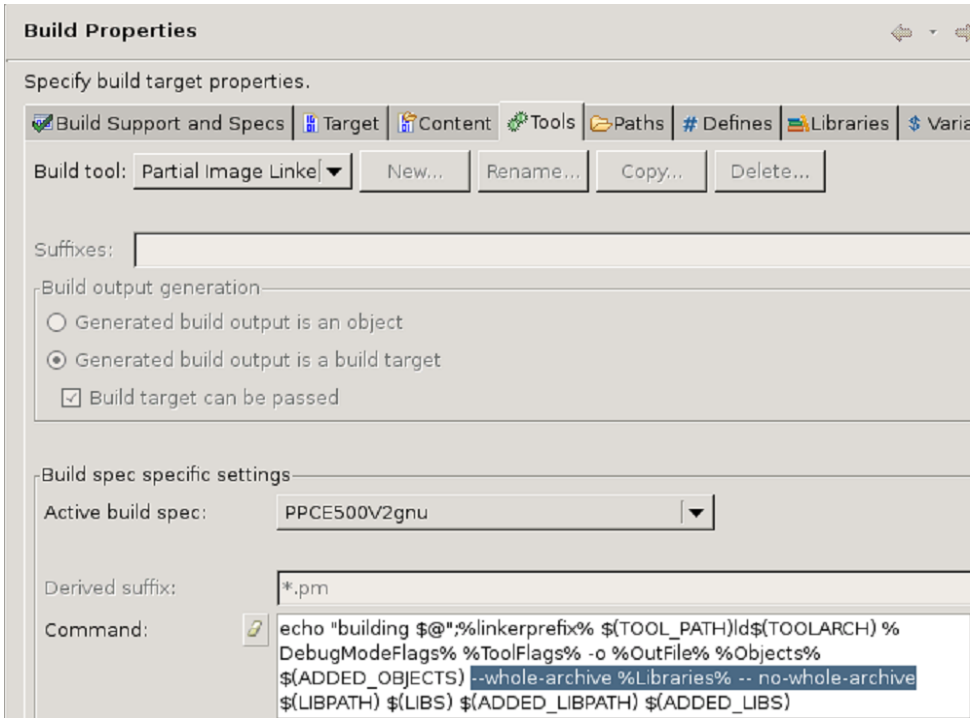
For C++:

```
$(WIND_BASE)/target/vThreads/lib/objPPCE6500gnuvx/vThreadsCplusComponent.o
$(WIND_BASE)/target/vThreads/lib/objPPCE6500gnuvx/vThreadsCplusLibraryComponent.o
$(WIND_BASE)/target/vThreads/lib/objPPCE6500gnuvx/vThreadsLocaleComponent.o
$(WIND_BASE)/target/vThreads/lib/objPPCE6500gnuvx/__ctype_tab.o
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddscore.so
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddsc.so
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddscpp.so
```

For C:

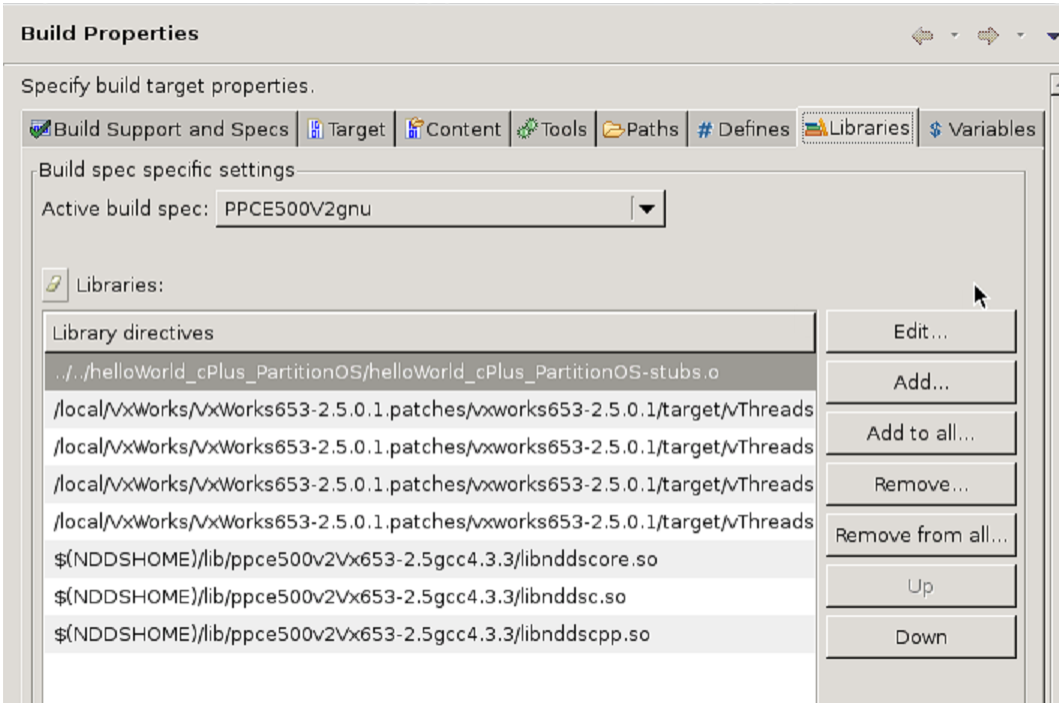
```
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddscore.so
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddsc.so
```

If you used RTI's *static* libraries (**rtiddscorez.a**, **rtiddscz.a**, and/or **rtiddscppz.a**), make sure to add this option to the linker command in the Tools tab within the Build Properties of your partitions: "**--whole-archive %Libraries% --no-whole-archive**". You can see an example in the following image:



g. Click **OK**.

For C++, it should look like this:

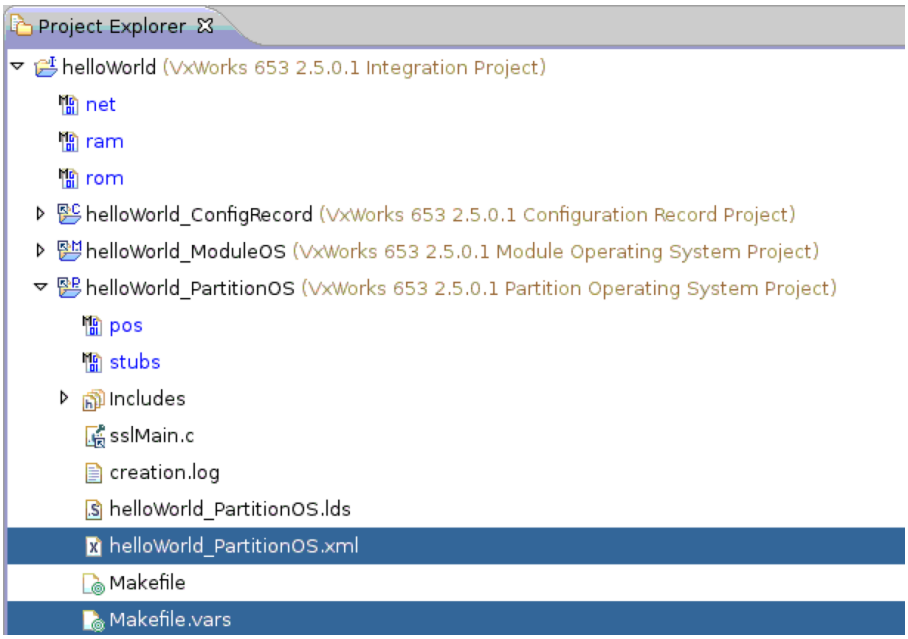


h. Repeat the same process for **helloWorld_subscriber**.

9. Build the Integration Project.

10. Add the POSIX interfaces and objects to the partitionOS.

    a. If you want to use POSIX API calls, you need to modify the following two files: **helloWorld_PartitionOS.xml** and **Makefile.vars** from the partitionOS project.



b. The XML file will look like this:

helloWorld_PartitionOS.xml

```
<Shared_Library_API
    xmlns="http://www.windriver.com/vxWorks653/SharedLibraryAPI"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    Name="vThreads"
    >

    <Interface>
    <Version Name="template"/>
    <xi:include href="$(WIND_BASE)/target/vThreads/config/comps/xml/vthreads.xml"/>
    <xi:include href="$(WIND_BASE)/target/vThreads/config/comps/xml/posix.xml"/>
    </Interface>

</Shared_Library_API>
```

c. The **Makefile.vars** file will look like this:

Makefile.vars

```
# Wind River Workbench makefile for partition operating system projects.

SSL_NAME = helloWorld_PartitionOS
CERT = 0
BSP = fsl_b4860_qds_AMP_CORE0
CPU = PPCE6500
API_FILE = $(SSL_NAME).xml
LDS_FILE = $(SSL_NAME).lds
XML_FILE = ../helloWorld_ConfigRecord/fsl_b4860_qds_AMP_CORE0_default.xml
BLACKBOX = vxSysLib
SSL_OBJS = $(filter-out $(SSL_NAME)-ept.o $(SSL_NAME)-stubs.o,$(patsubst %.c,%.o,$(wil
SSL_OBJS += vThreadsComponent.o vThreadsPosixInit.o vThreadsPosixComponent.o
```

6.2 Running Connext DDS Applications for VxWorks 653 2.5.x

1. Boot up your target board with the kernel created by the Integration project.
2. If the *Connext DDS* applications are in schedule 0, they will start up automatically, and you should see the publisher and subscriber communicating with each other.
3. If the *Connext DDS* applications are not in schedule 0, use this command to change to the desired schedule: **arincSchedSet <Schedule number>**.

Chapter 7 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 Step 1, Set up the Environment, in the RTI Connext DDS Core Libraries 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 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.)

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 as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries 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
   ```

   Edit the generated example code as described in Generating Code with RTI Code Generator, in the RTI Connext DDS Core Libraries Getting Started Guide.
4. Set up your environment with the **wrenv.sh** script in the Wind River Linux base directory.

   ```
   wrenv.sh -p 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, 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...
   ```
7. Connect to the Wind River Linux 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
   ```
8. Look back in the first (subscribing) shell. 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...
```