# RTI Connext DDS

# Core Libraries

# Getting Started Guide

# Addendum for Embedded Systems

## Version 5.3.0

Printed in U.S.A. First printing.
June 2017.

**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 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 Generating Code with RTI Code Generator, 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. Please refer to the documentation provided by Green Hills Systems for more information about this operating system.

This process has been tested on INTEGRITY 5.0.11 and assumes that applications are downloaded dynamically.

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 Building the Kernel (Section 3.1 below).

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


- Generate Example Code and Project File with rtiddsgen (Section 3.2.1 on the facing page)
- Build the Publish and Subscribe Applications (Section 3.2.2 on the facing page)
- Connect to the INTEGRITY Target from MULTI (Section 3.2.3 on page 7)
- Load the Application on the Target (Section 3.2.4 on page 7)
- Run the Application and View the Output (Section 3.2.5 on page 8)

### 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 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 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 Systems

This section provides simple instructions to configure a kernel and run Connext DDS applications on VxWorks 6.x 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 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 section describes how to build the kernel:

- Building the Kernel (Section 4.1 below)

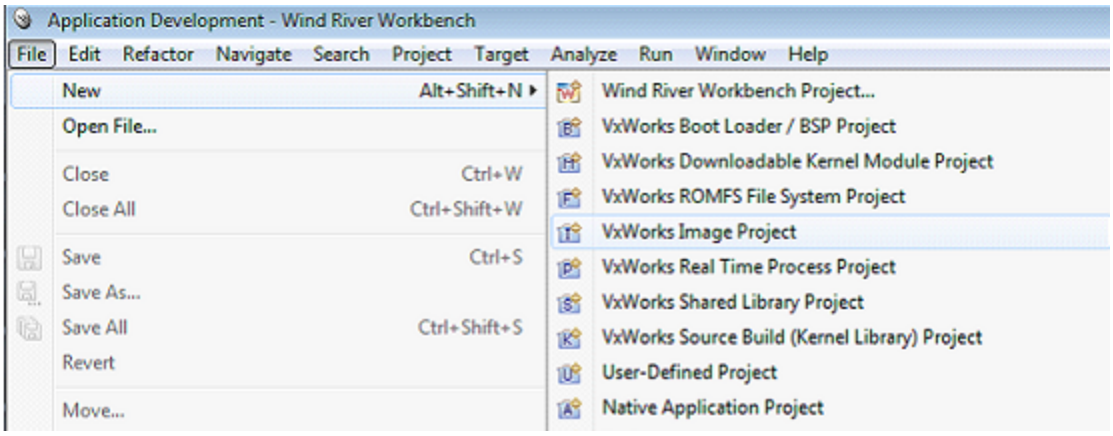
The next section guides you through the steps to generate, modify, build, and run the provided example HelloWorld application on a VxWorks target:

- Building and Running a Hello World Example (Section 4.2 on page 15)

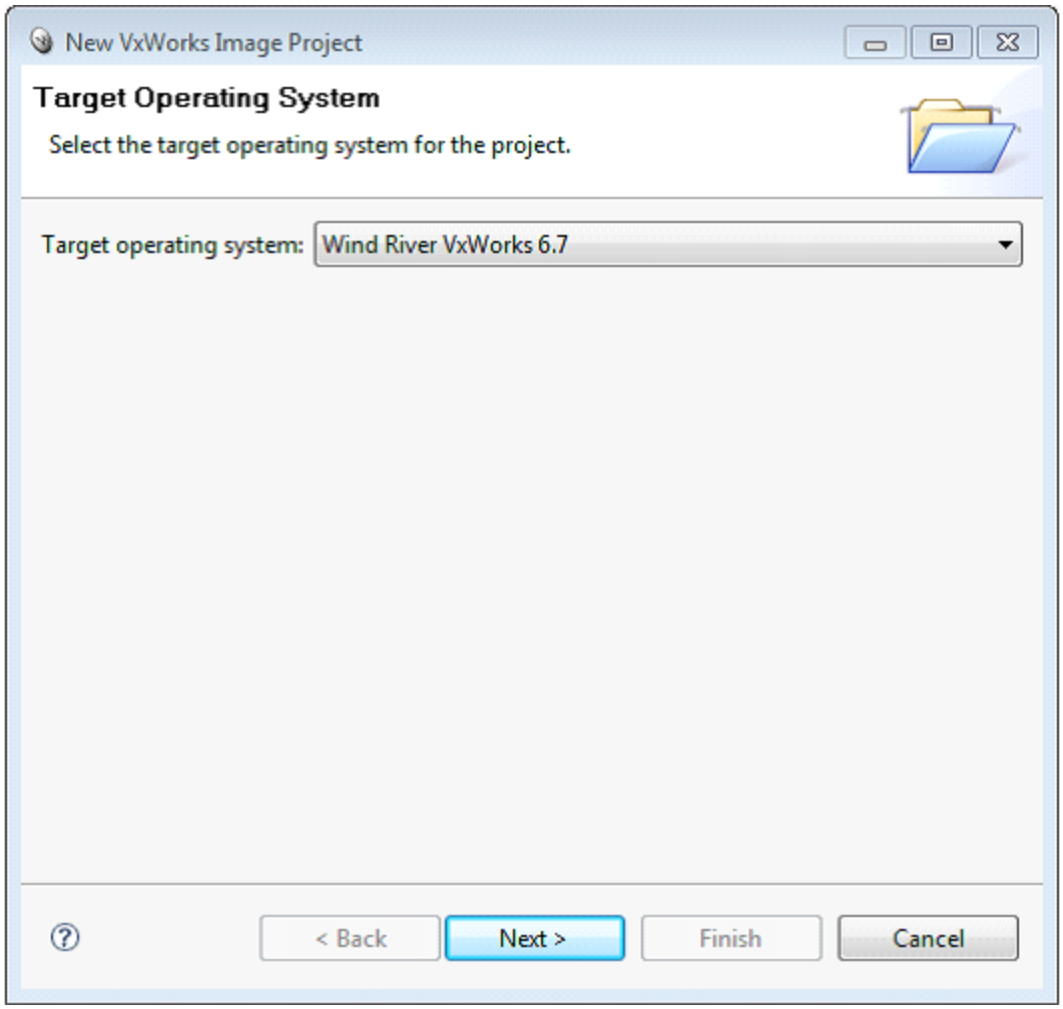
## 4.1 Building the Kernel

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

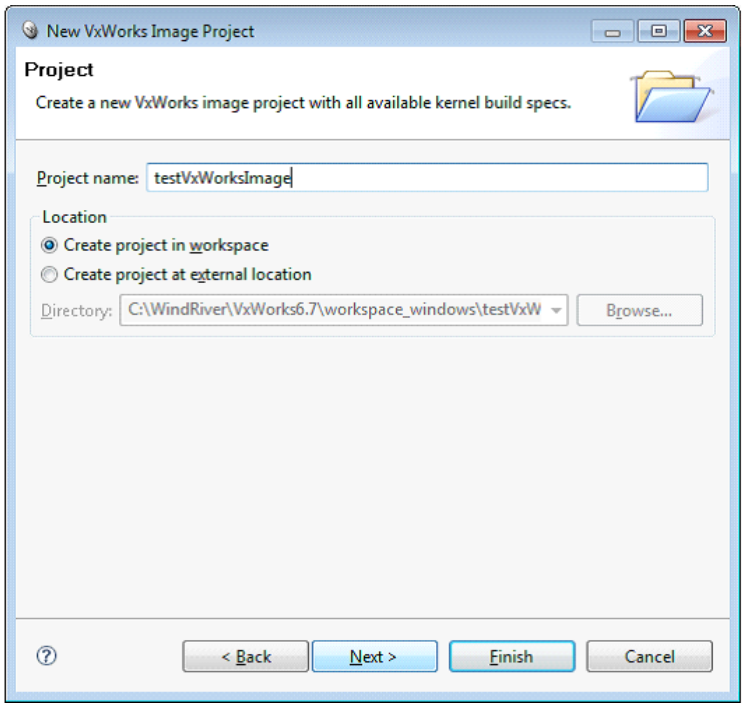
1. Launch Workbench.
2. Select **File, New, VxWorks Image Project**



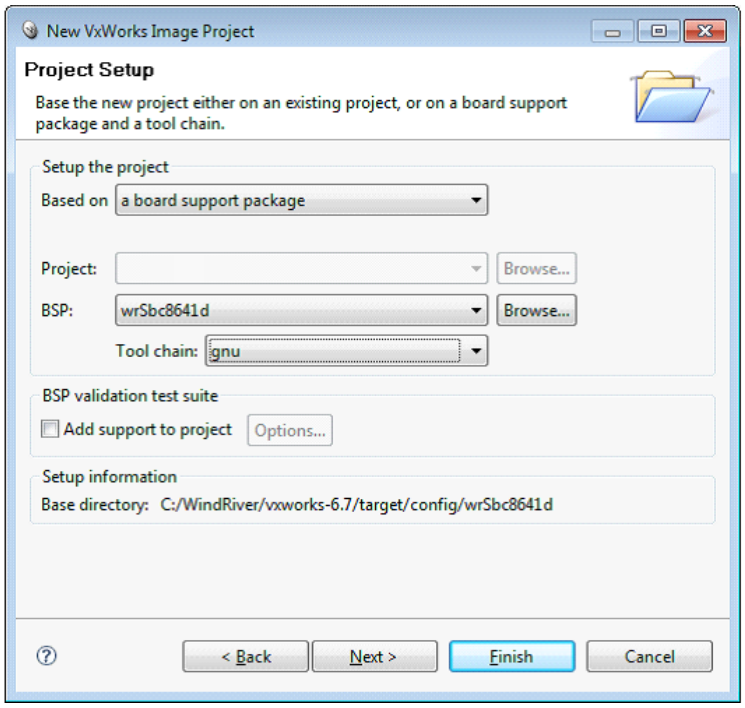

3. Select the desired operating system; click **Next**.



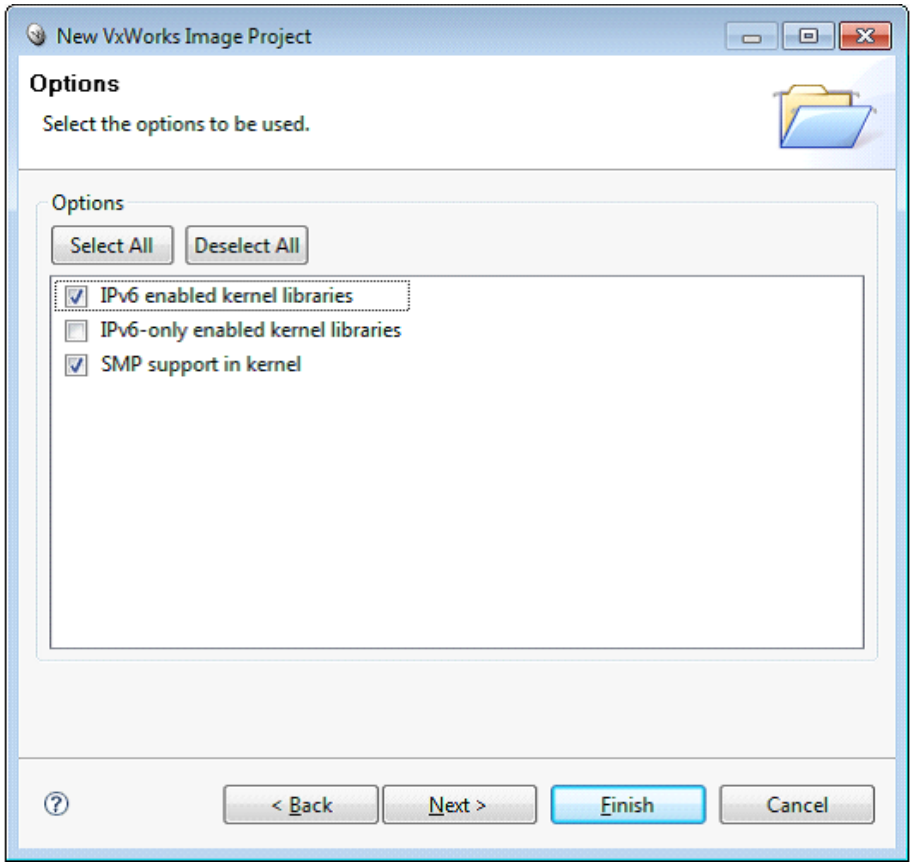

4. Give your project a name; click **Next**.



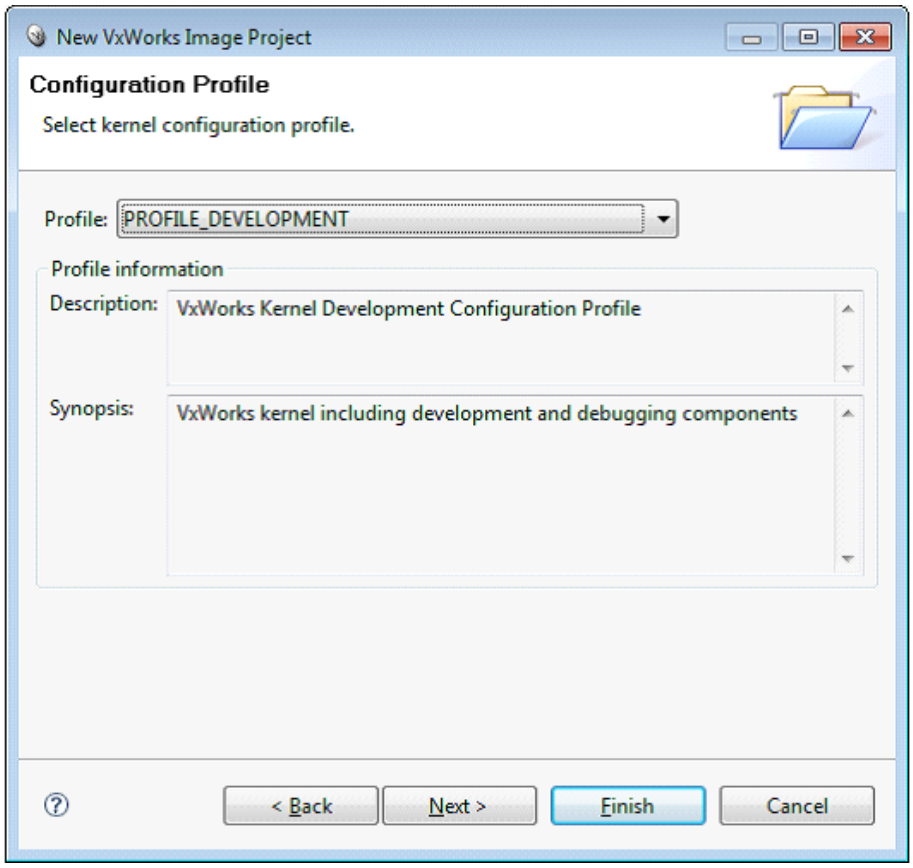

5. Choose the **board support package (BSP)** based on your hardware.
6. For VxWorks 6.9: Select the correct **Address mode**.
7. For the Tool chain option, select **GNU**; click **Next**.



8. Select SMP support if your BSP supports it and you want to enable symmetric multi-processing capability in the kernel. To see if your architecture supports IPv6, consult the *Platform Notes*.



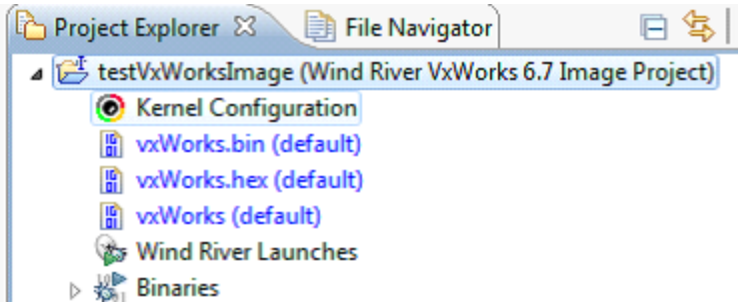

9. Select **PROFILE_DEVELOPMENT**.




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

    Your project will be created at this time.

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




12. Either:

    a. For VxWorks 6.8 and higher:

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

    b. For VxWorks 6.7, ONLY if you have enabled SMP support in the kernel:

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

13. 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*.

14. 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.

15. 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.

16. For VxWorks 6.4 and below, add the following modules:

    - **ZBUF Socket** (under Network Components, Network Socket Components)

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

      This will enable multicast for the target.

17. 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).

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.

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/**.

## 4.2 Building and Running a Hello World Example

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

### 4.2.1 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 the Connext DDS (*rtiddsgen*) utility 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.

### 4.2.2 Building and Running an Application as a Kernel Task

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

### 4.2.2.1 Using the Command Line

1. Set up your environment with the **wrenv.sh** script or **wrenv.bat** batch file in the VxWorks base directory.
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
   ```

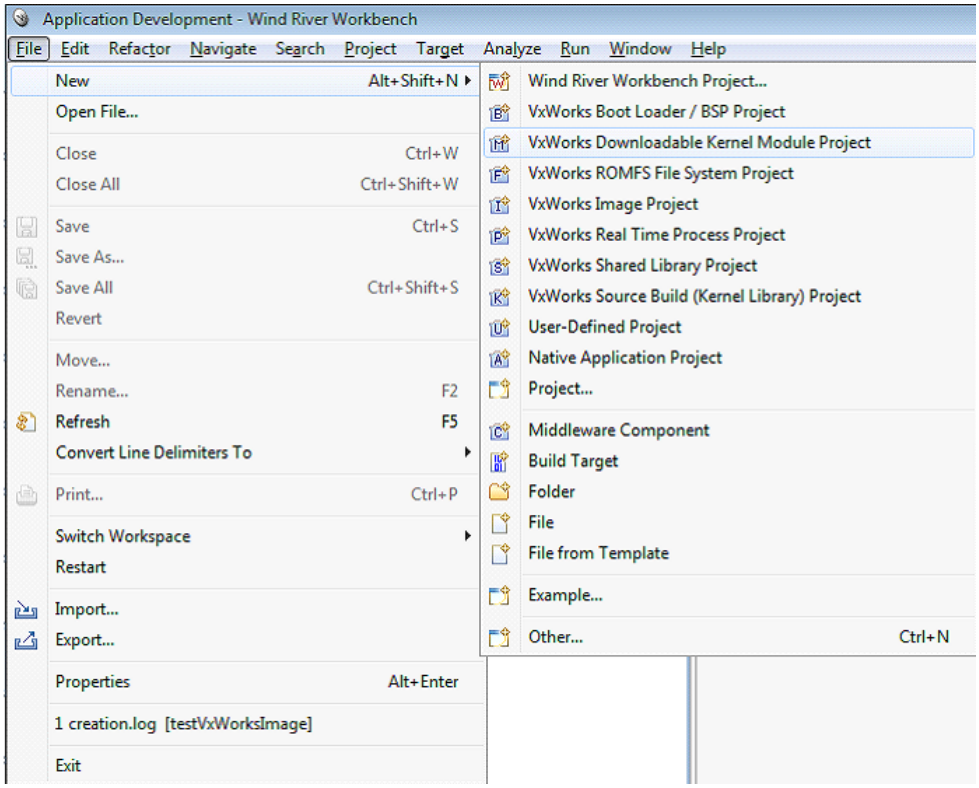
   (Where 'directory' refers to the location of the generated object files.) If you are using dynamic linking: load the libraries first, in this order: **libnddscore.so**, **libnddsc.so**, **libnddscpp.so**; *then* load the **.so** file produced by the build.
9. Run the **subscriber_main** or **publisher_main**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

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

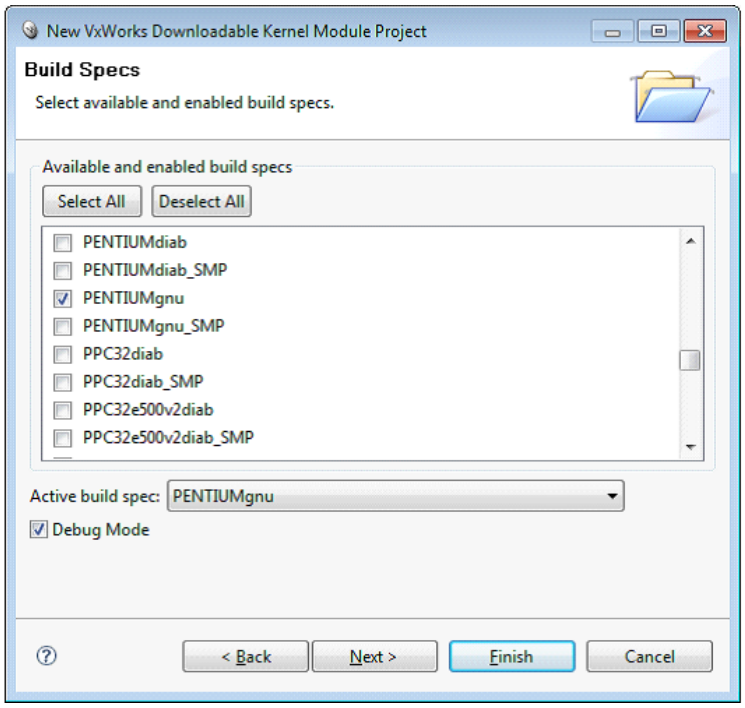
In this example, 38 is the domain ID and 10 is the number of samples.

### 4.2.2.2 Using Workbench

1. Start Workbench.
2. Select **File, New, VxWorks Downloadable Kernel Module Project**.

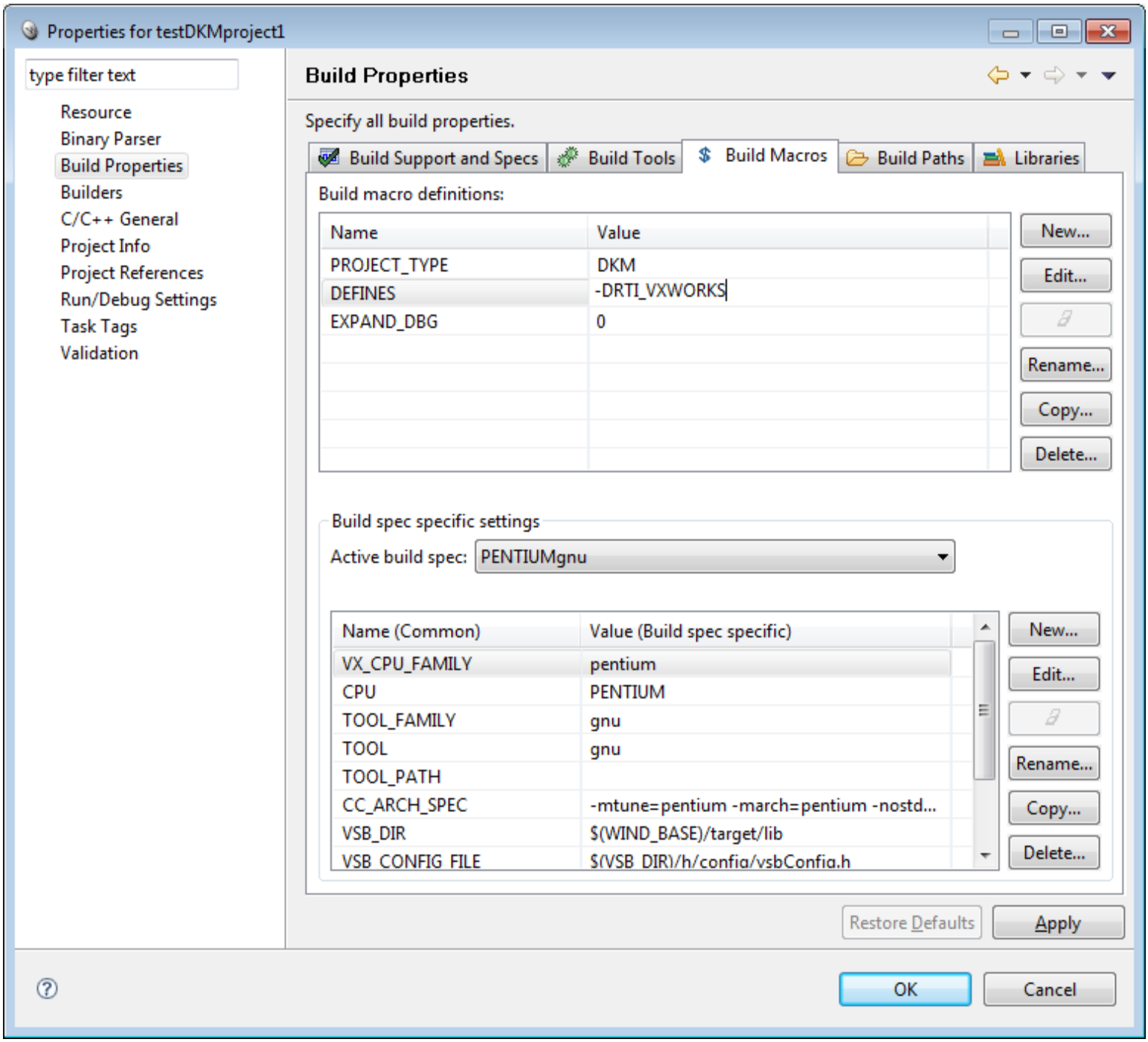



3. Give your project a name; click **Next**.
4. Select the default options until you reach the dialog titled **Build Specs**. In this dialog, choose the desired build spec.




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

   Your project will be created at this time.
6. Copy the source files and headers generated by *rtiddsgen* in Generate Example Code and Makefile with rtiddsgen (Section 4.2.1 on page 15) into the project directory.
7. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.
8. Open the project Properties by right-clicking on the project in Project Explorer and selecting **Properties**.
9. In the dialog box that appears, select **Build Properties** in the navigation pane on the left.




10. In the Build Macros tab:

    Add **-DRTI_VXWORKS** to DEFINES in the Build macro definitions.

    If you are using static linking, in the Variables tab:

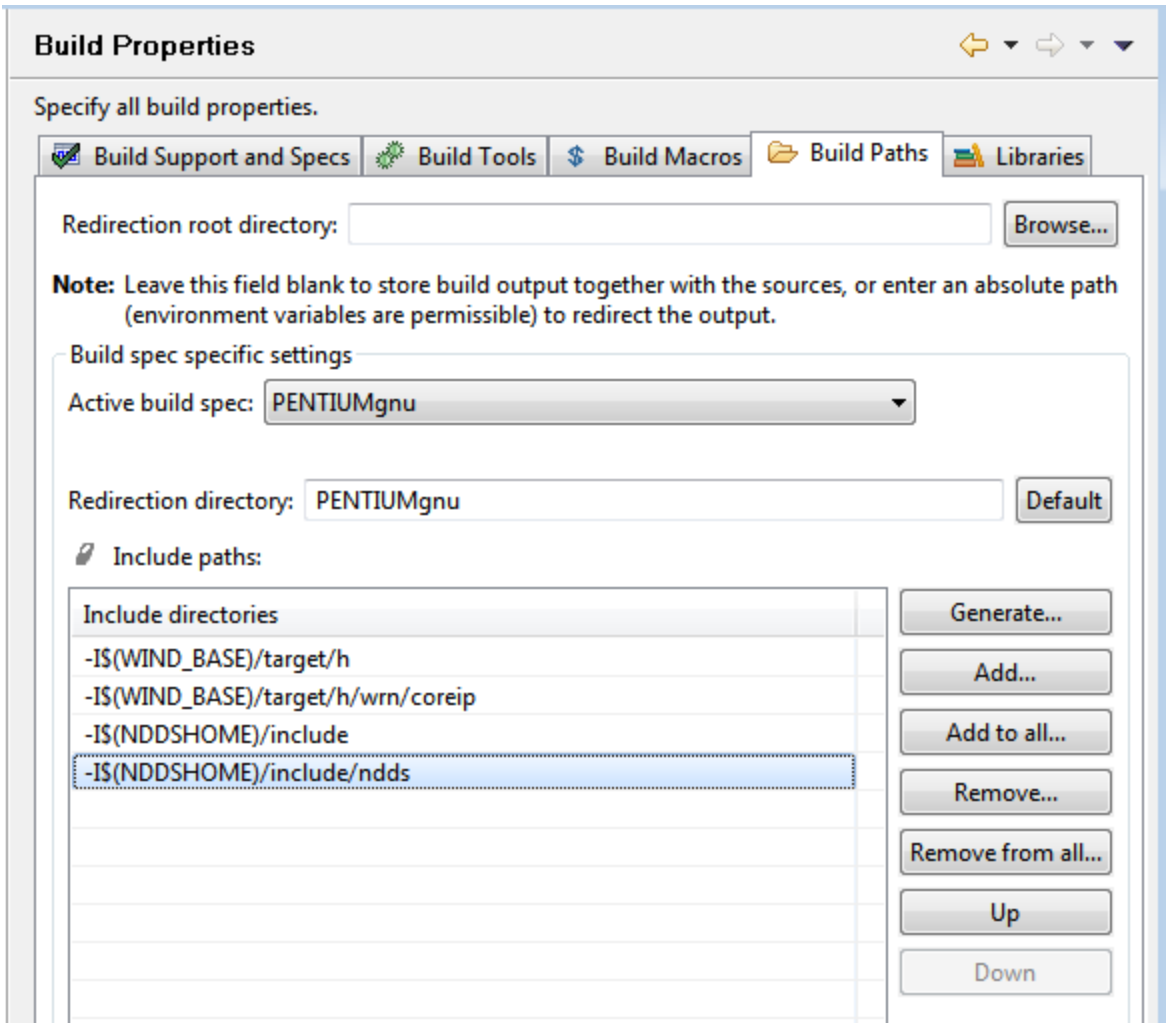
- Add to LIBPATH: **-L$(NDDSHOME)/lib/<*architecture*>**
- Add to LIBS: **-lnddscppz -lnddscz -lnddscorez** *(in that order)*

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

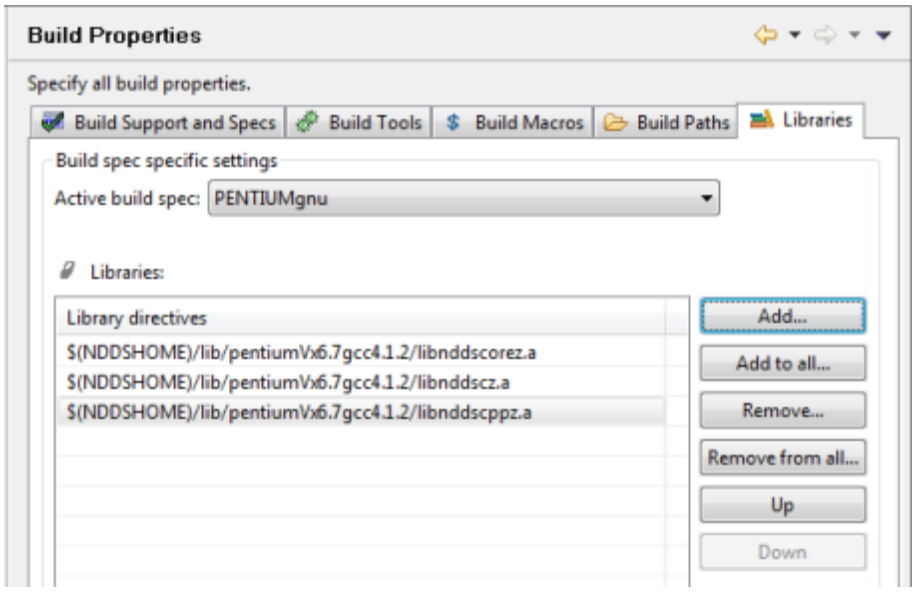
11. In the **Build Paths** tab, add both of these:

    **-I$(NDDSHOME)/include**

    **-I$(NDDSHOME)/include/ndds**




12. If you are using dynamic linking: In the Libraries tab, add the Library directives shown below:




13. Click **Apply** to save the changes, then click **OK** to exit the Properties menu.
14. Build the project by right-clicking on the project in Project Explorer, then selecting **Build**.
15. 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.

## 4.2.3 Building and Running an Application as a Real-Time Process

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

- Using the Command Line (Section 4.2.3.1 below)
- Using Workbench (Section 4.2.3.2 on the next page)

### 4.2.3.1 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.
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 (VxWorks 6.3 and above only), make the following modifications the generated makefile:

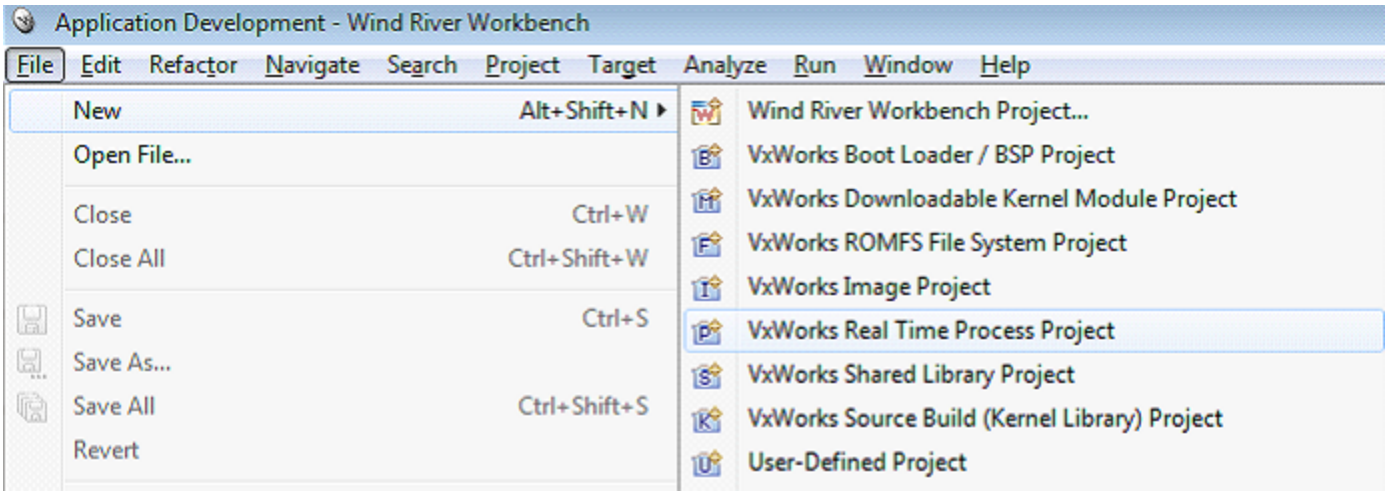
```
LIBS = -L$(NDDSHOME)/lib/<architecture> -non-static -lnddscpp \-
lnddsc -lnddscore $(syslibs_<architecture>)
```

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** sub-menu.
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**.

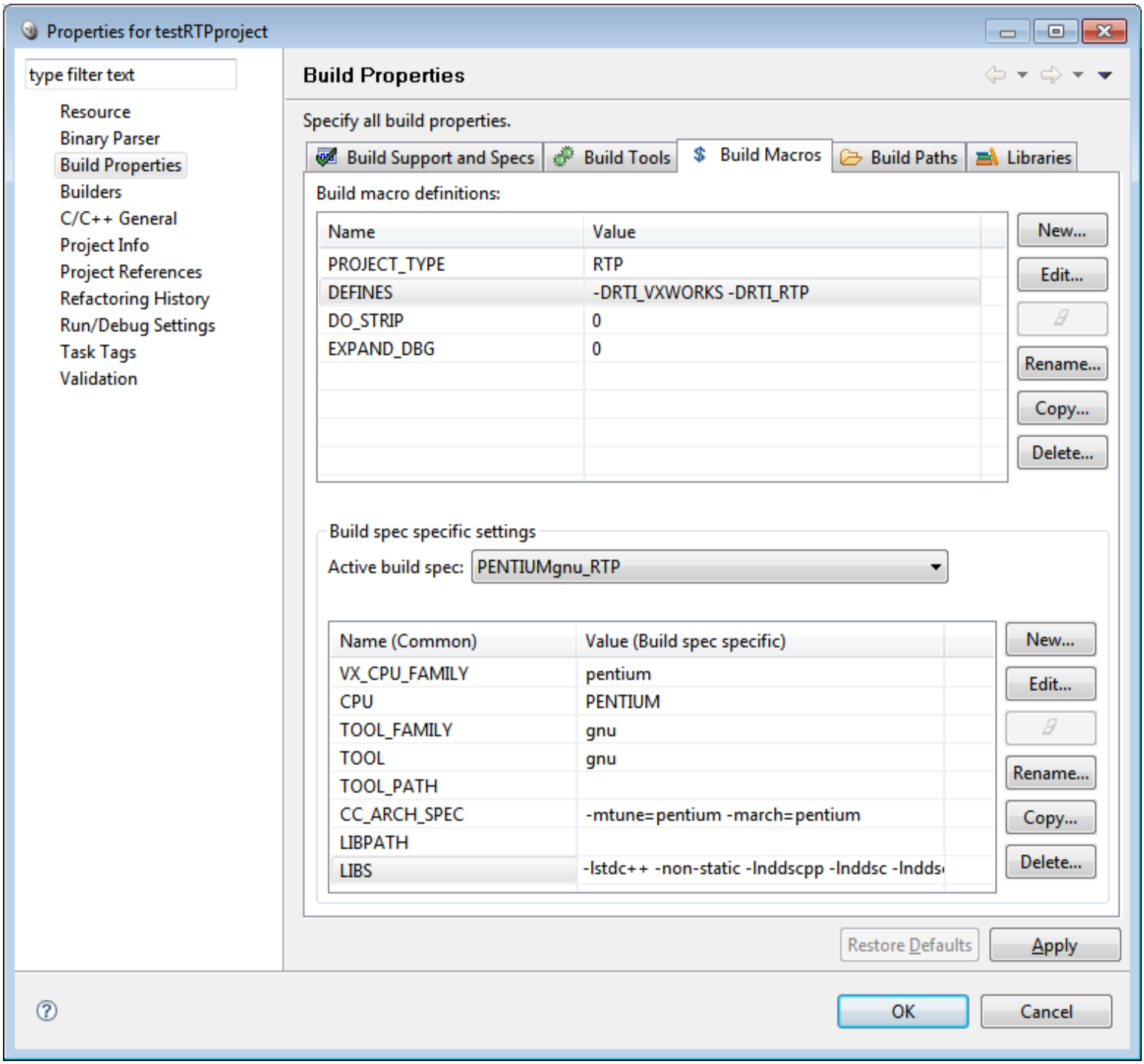
### 4.2.3.2 Using Workbench

1. Start Workbench.
2. Select **File, New, VxWorks Real Time Process Project**.
3. Give your project a name; click **Next**.
4. You can select the default options until you reach the dialog titled **Build Specs**. In this dialog, choose the desired build spec:




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

   Your project will be created at this time.
6. Copy the source and header files generated by *rtiddsgen* in Generate Example Code and Makefile with rtiddsgen (Section 4.2.1 on page 15) 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.
7. View the added files by right-clicking on the project in Project Explorer, then selecting **Refresh** to see the files.
8. Open the project Properties by right-clicking on the project in Project Explorer and selecting Properties.
9. In the dialog box that appears, select **Build Properties** in the navigation pane on the left.




10. In the Build Macros tab: Add **-DRTI_VXWORKS -DRTI_RTP** to DEFINES in the Build macro definitions.

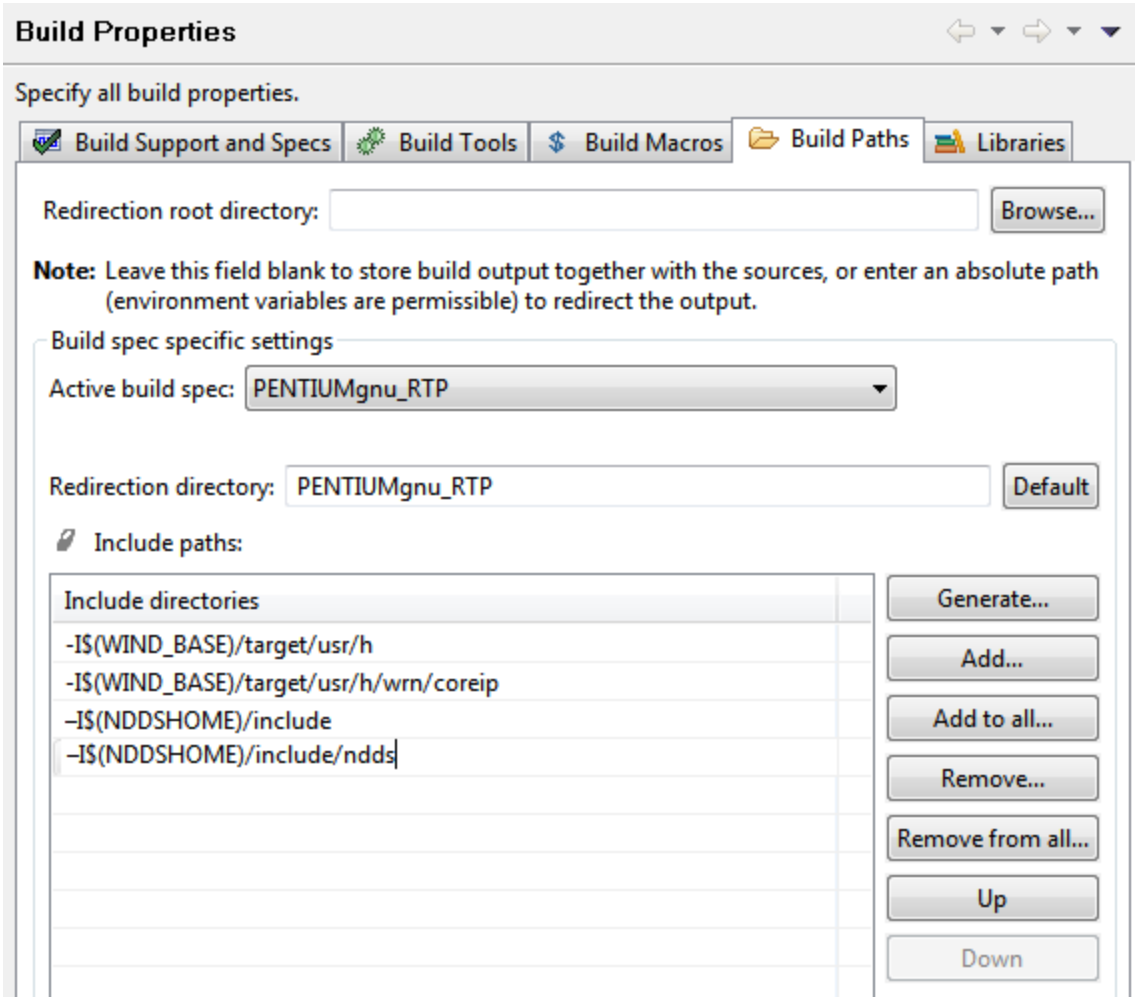
    If you are using **static** linking, in the Variables tab:

- Add to LIBPATH: **-L/(NDDSHOME)/lib/<*architecture*>**
- Add to LIBS: **-lnddscppz -lnddscz -lnddscorez** *(in that order)*

If you are using *dynamic* linking, in the Variables tab:

- Add to LIBS: **-non-static -lnddscpp -lnddsc -lnddscore** *(in that order)*

11. In the Build Paths tab, add:
    - **-I$(NDDSHOME)/include**
    - **-I$(NDDSHOME)/include/ndds**




12. Click **Apply** to save the changes, then click **OK** to exit the Properties menu.
13. Build the project, by right-clicking on the project in Project Explorer, then selecting **Build**.
14. Run the application as described starting in Step 5 in the Command Line section above.

# 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. (Section on page 37) through Configure properties for the application. (Section on page 38)

**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 Creating Connext DDS Applications for VxWorks 653 v2.3 Platforms (Section 5.2 on the facing page).

**For someone creating a Connext DDS application**, this section provides an example from the ground up.

## 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:

Installing the Wind River Services Socket Library (Section 5.1.1 below)

or

Installing the RTI Socket Library (Section 5.1.2 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

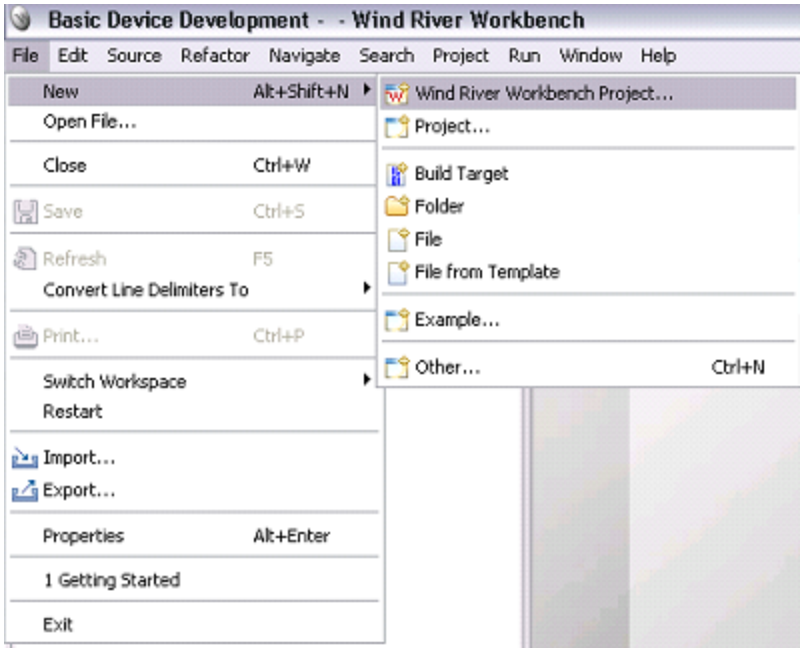
1. Install Workbench.
2. Install **vx_653_socket.<Connext DDS version>**.
   a. Copy the socket driver files from RTI to each BSP of interest. Once you extract the RTI Socket Library zip file into your <NDDSHOME> installation directory, copy the contents of **vx_653_socket.<Connext DDS version>\bsp\src**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

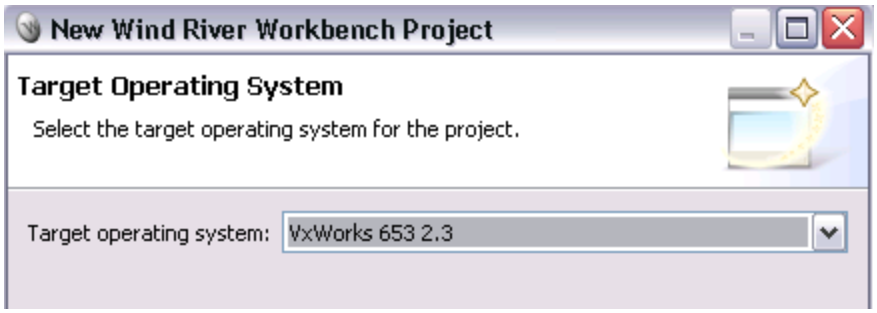
## 5.2 Creating Connext DDS Applications for VxWorks 653 v2.3 Platforms

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.

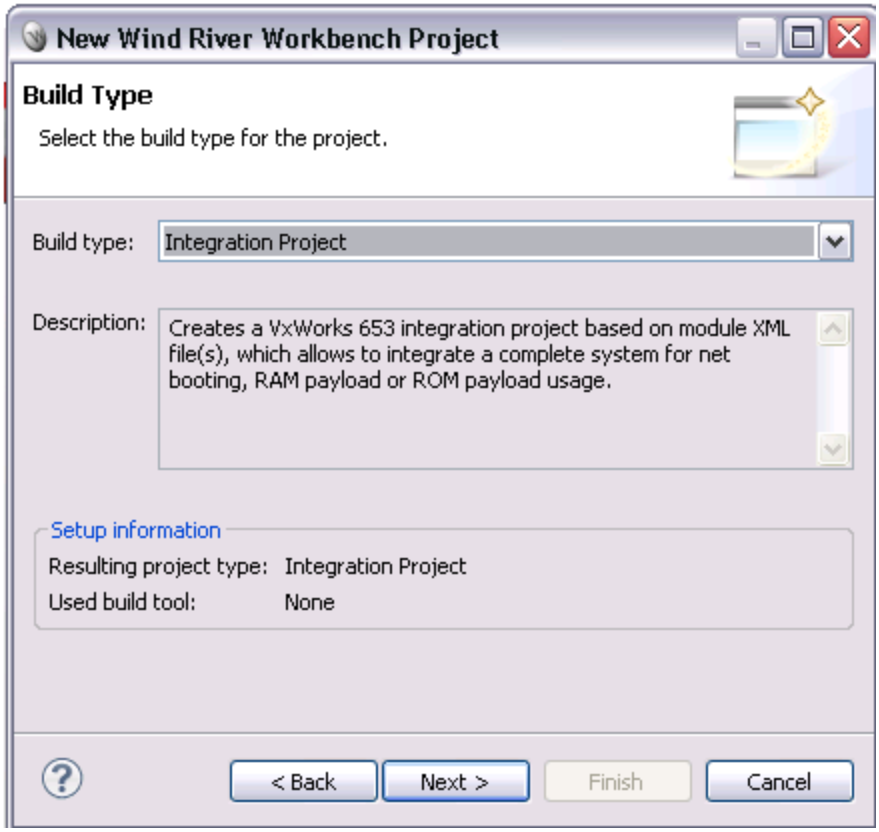
1. Create an integration project with two partitions (one for the publisher, one for the subscriber). Follow the instructions from Wind River for doing this. The following screenshots will guide you through the process.
   a. Create a new Workbench project.



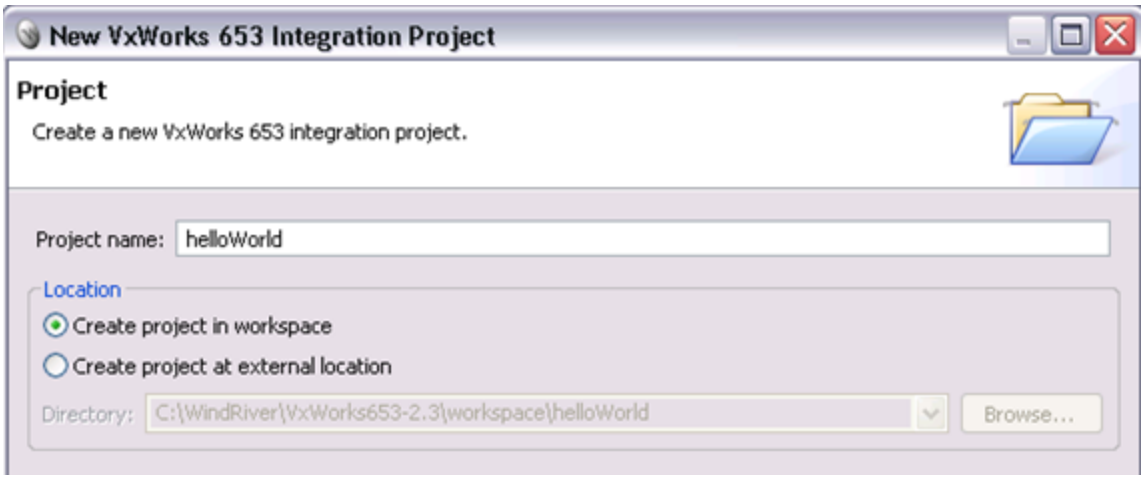

   b. For the Target operating system, select **VxWorks 653 2.3**.



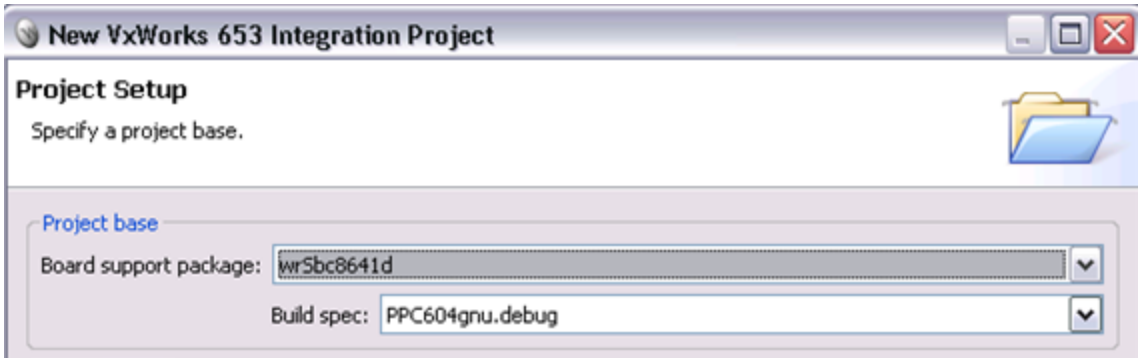

c. For Build type, select **Integration Project**.



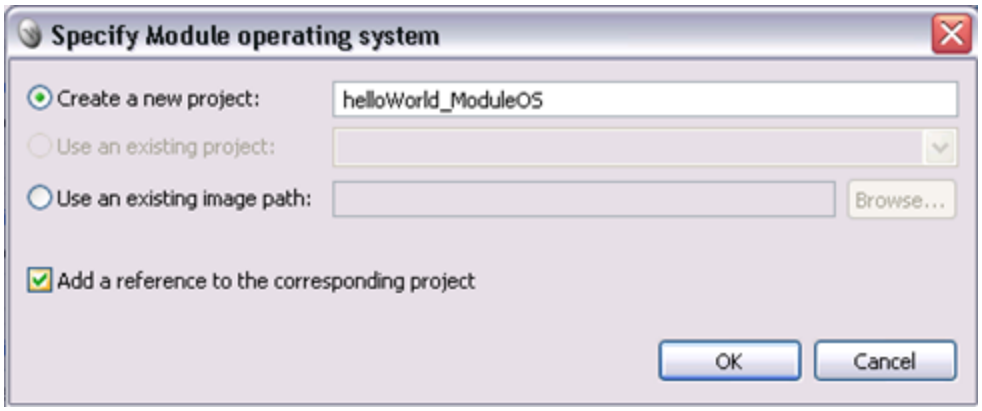

d. Create a project named **helloWorld** in the workspace.

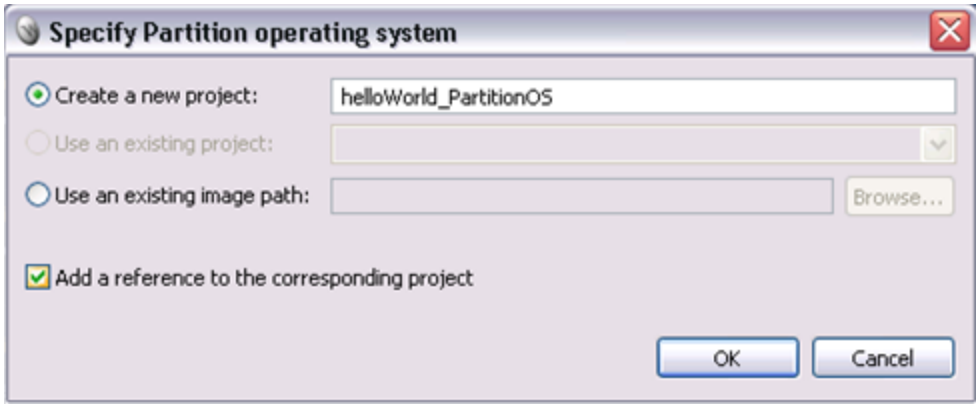



e. Select the appropriate Board Support package. Make sure the debug Build spec is selected. This example assumes the **wrSbc8641d** board support package is selected; alternatively, you could select **simpc**.

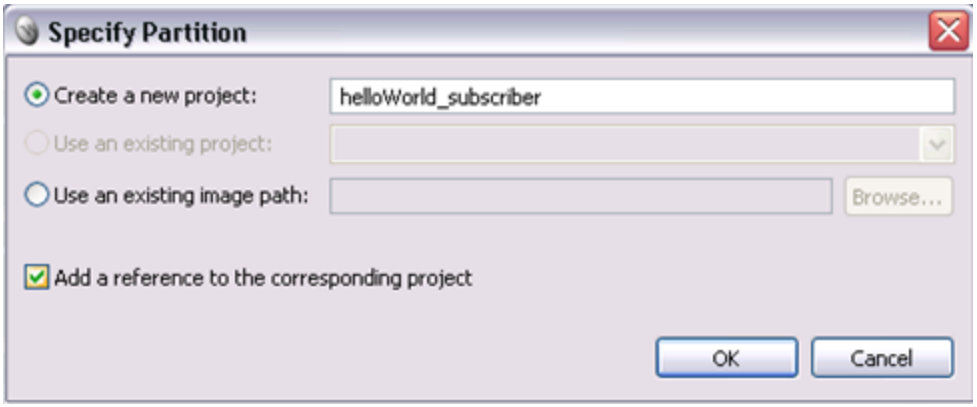



f. Select the default options for adding the ConfigRecord, ModuleOS, and PartitionOS. Make sure the "**Add a reference to the corresponding project**" checkbox is selected.

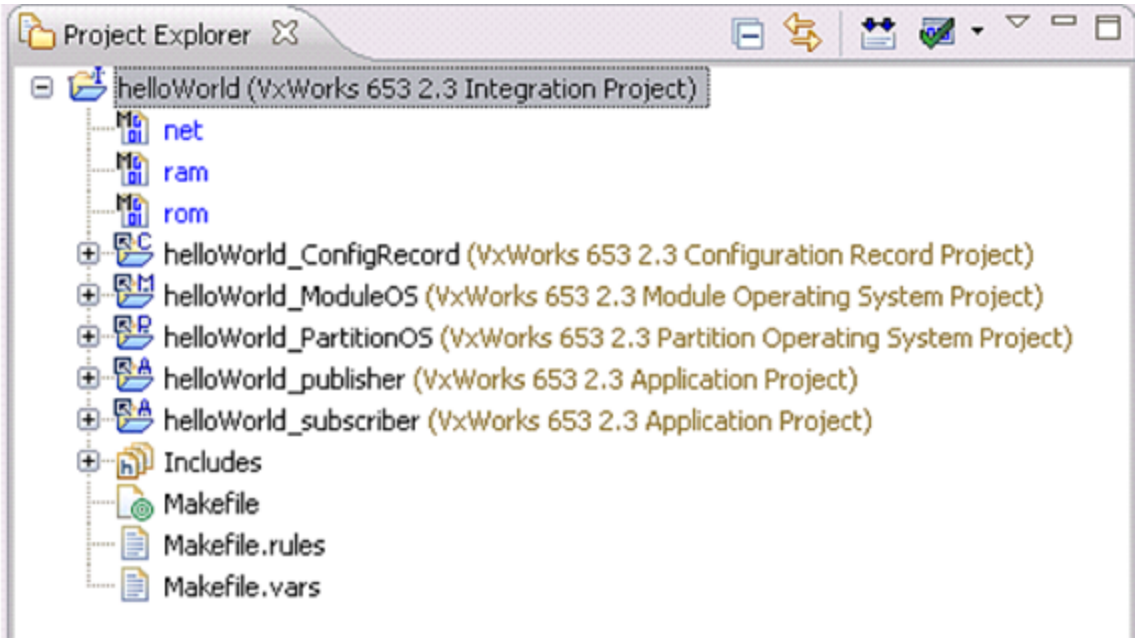



g. Create two partitions, **helloWorld_publisher** and **helloWorld_subscriber**, to create a Publisher and a Subscriber application, respectively. Make sure the **"Add a reference to the corresponding project"** checkbox is selected.

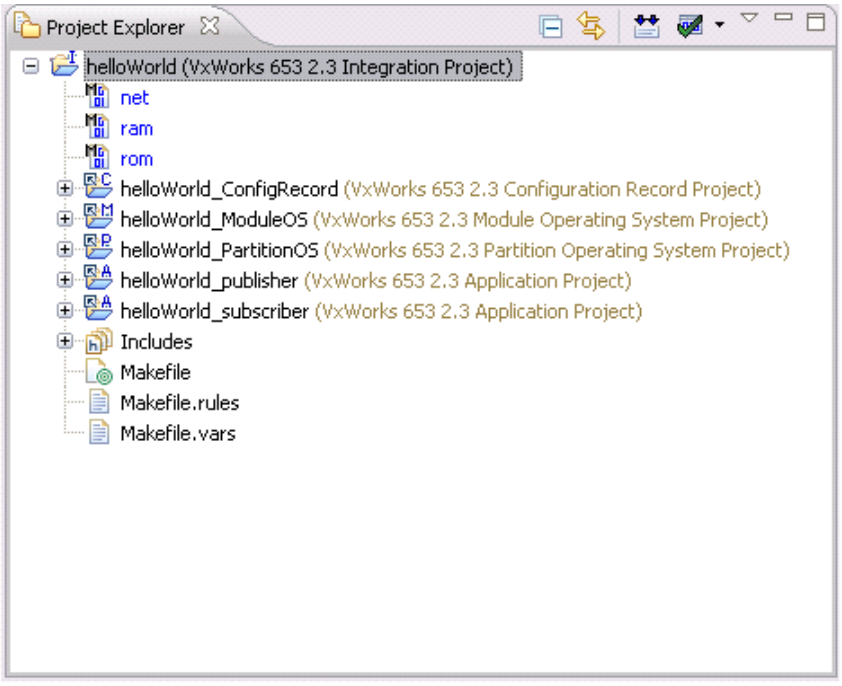



h. Now you are ready to create the Integration Project.




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 **hel-**

**loWorld_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]**.

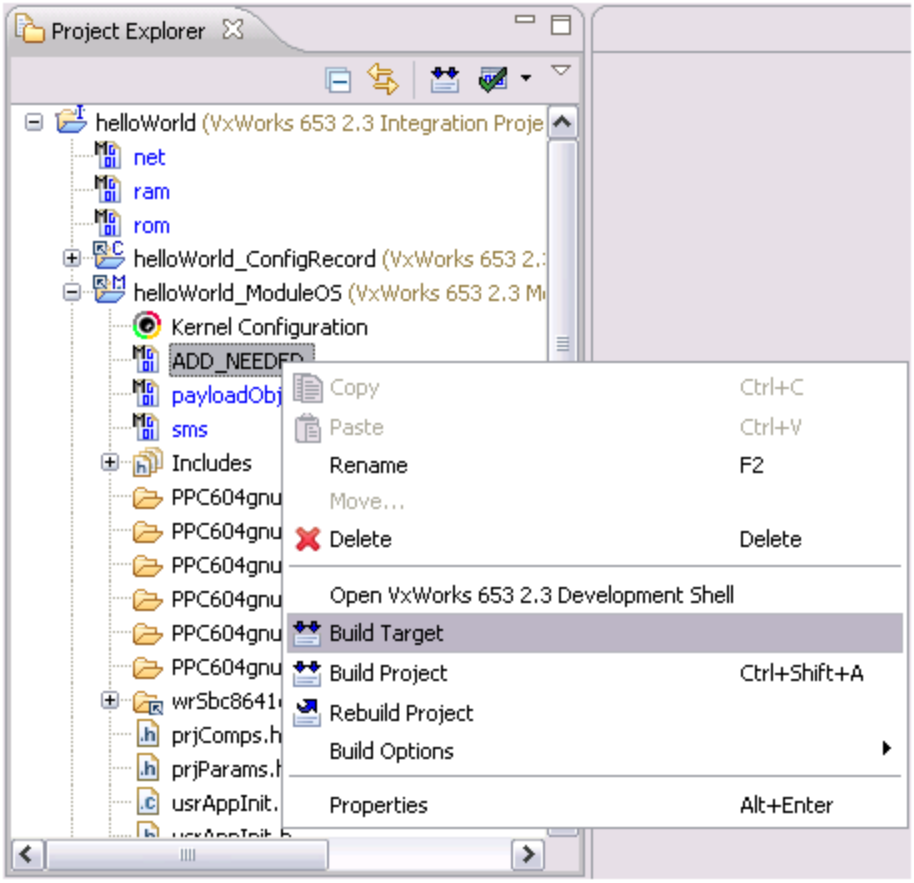
b. Include **development tool components->**
**debug utilities [INCLUDE_DEBUG_UTIL]**. This is needed to enable worker tasks.

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:

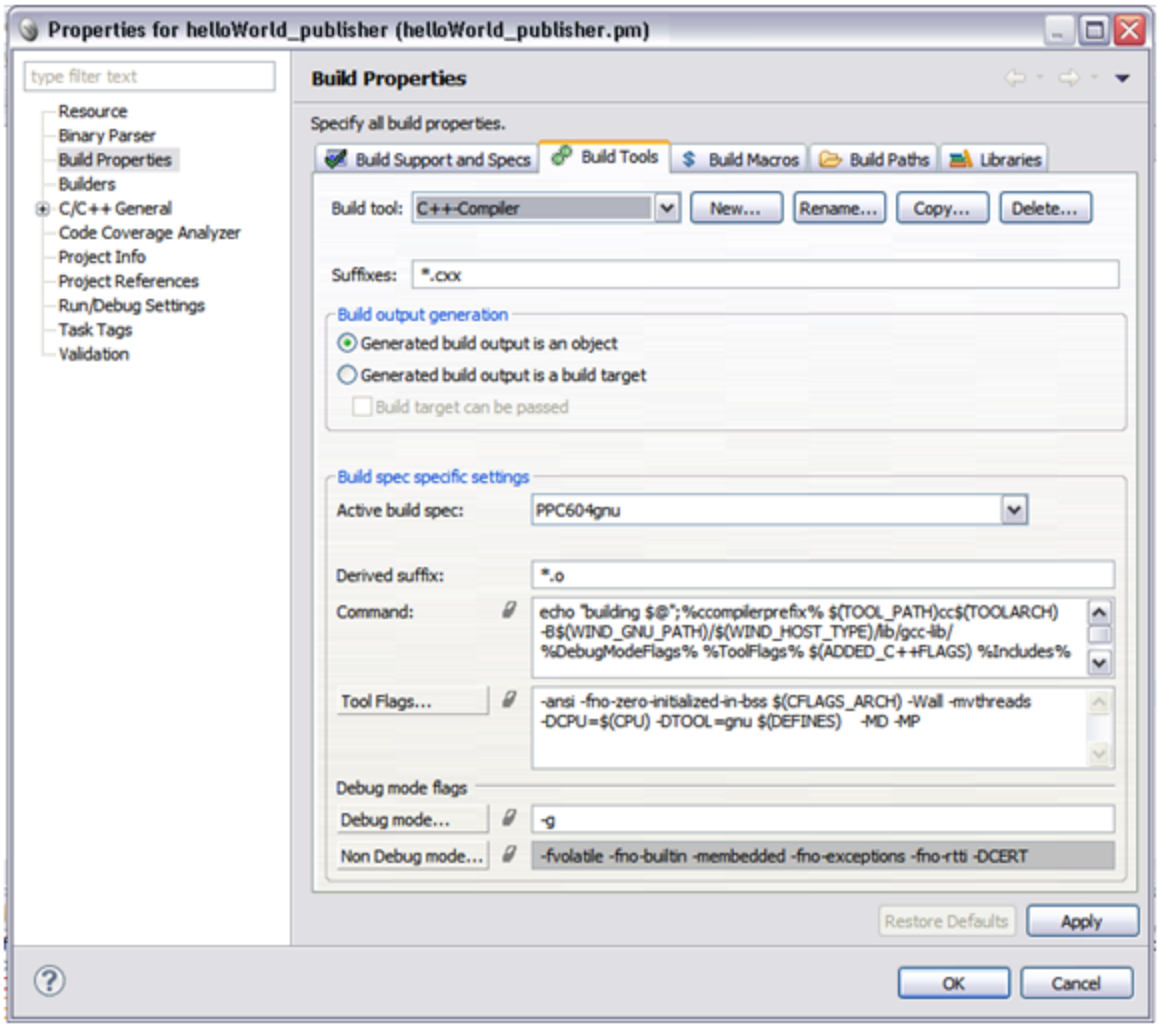
- Add a new macro, **NDDSHOME**, and set its value to the location where Connext DDS is installed. If this is in a directory with spaces in the path (such as Program Files), put quotation marks around the whole path. For the path, use forward slashes ("/"), not backslashes ("\").

- 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**.

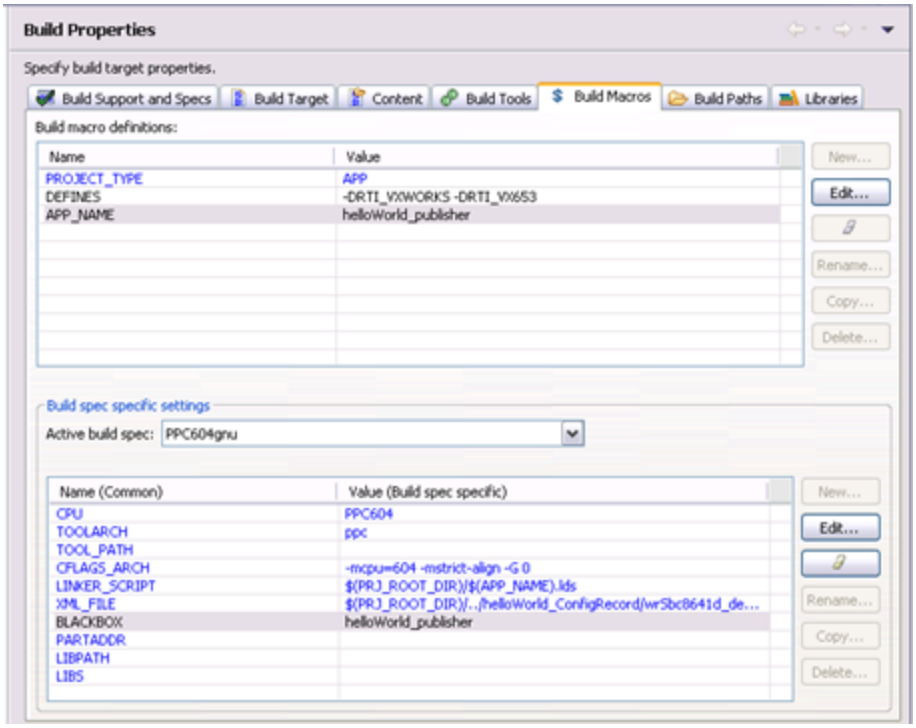



b. For C: Right-click **helloWorld_publisher**.

   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

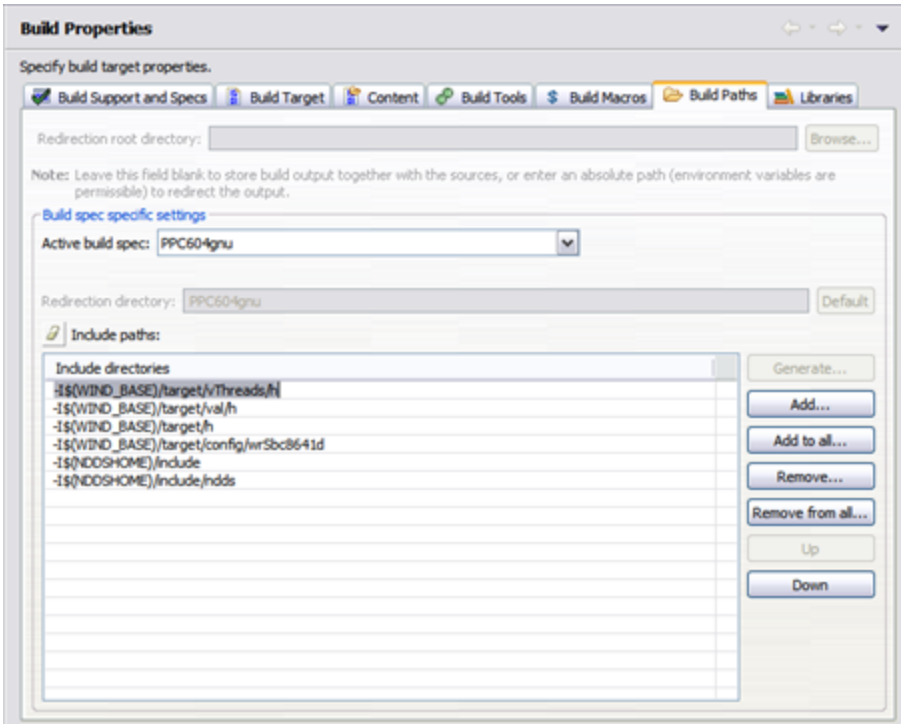
- **simpcVx653-2.3gcc3.3.2**

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

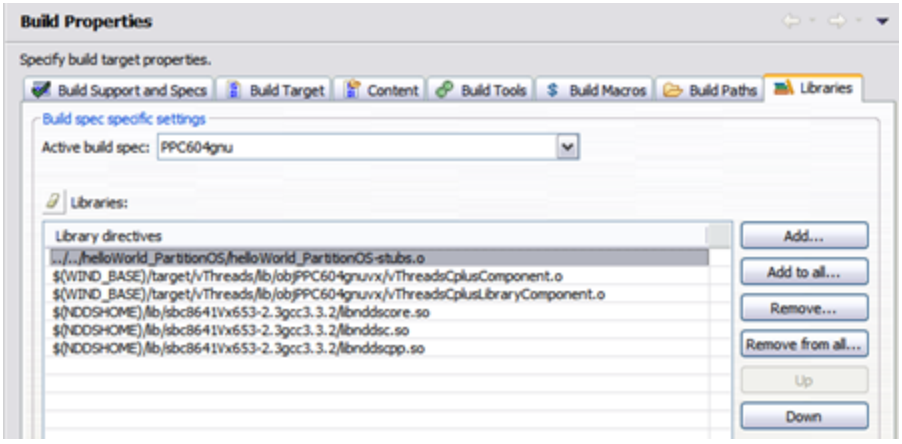
Make sure you have added the libraries as fully qualified names (without **-l** or **-L**).

*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 v2.5.0.1 Systems

This chapter provides simple instructions on how to configure a kernel and run Connext DDS applications on a VxWorks 653 version 2.5.0.1 system. 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 2.5.0.1 Platforms.*

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 Creating Connext DDS Applications for VxWorks 653

2.5.0.1 (Section 6.1 on the next page) (start with the step to Generate example code with rtiddsgen. (Section on page 53), through the step to Configure properties for the application. (Section on page 54)).

**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 Creating Connext DDS Applications for VxWorks 653 2.5.0.1 (Section 6.1 below).

**For someone creating a Connext DDS application**, this document provides an example from the ground up.

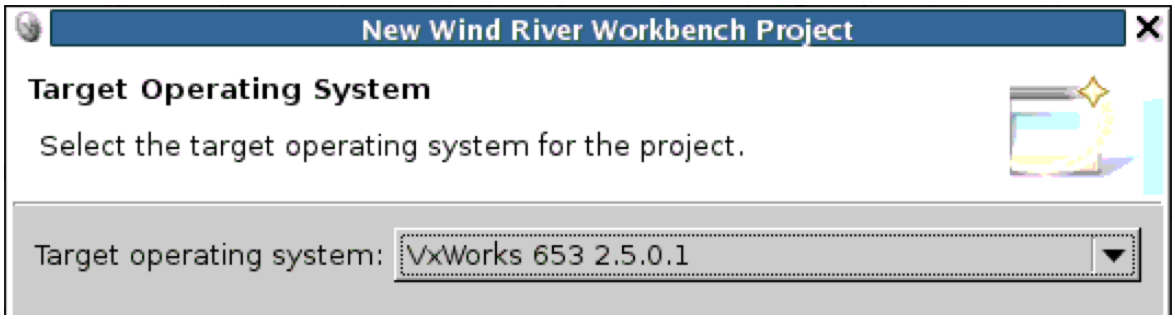
## 6.1 Creating Connext DDS Applications for VxWorks 653 2.5.0.1

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.

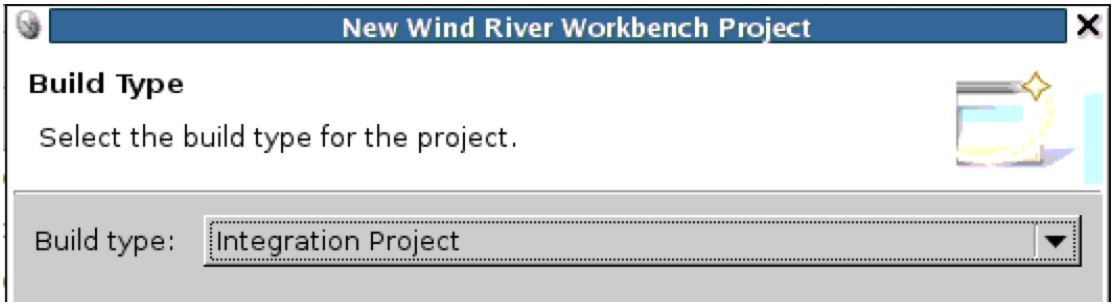
1. Create an integration project with two partitions (one for the publisher, one for the subscriber). Follow the instructions from Wind River for doing this. The following screenshots will guide you through the process.

    a. Create a new Workbench project.

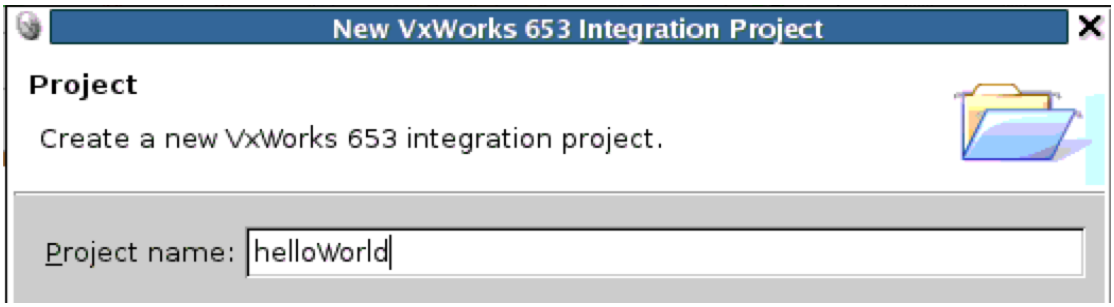

    b. For the Target operating system, select **VxWorks 653 2.5.0.1**.



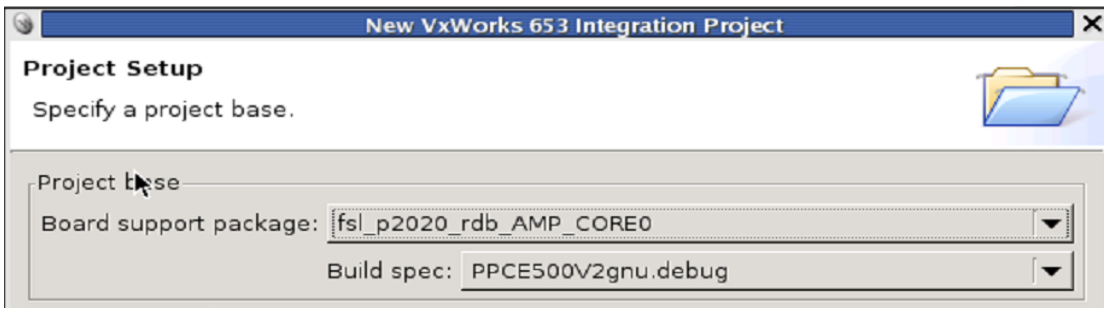
c. For Build type, select **Integration Project**.



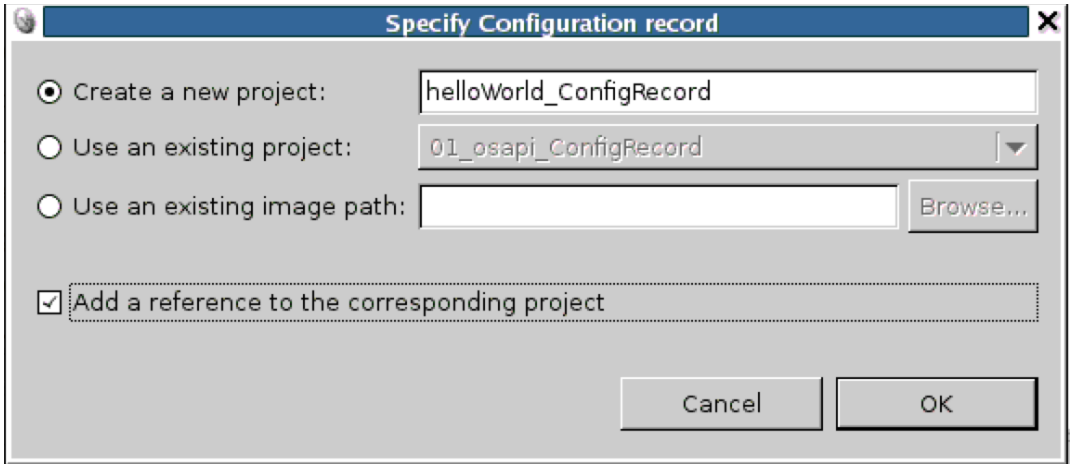

d. Create a project named **helloWorld** in the workspace.

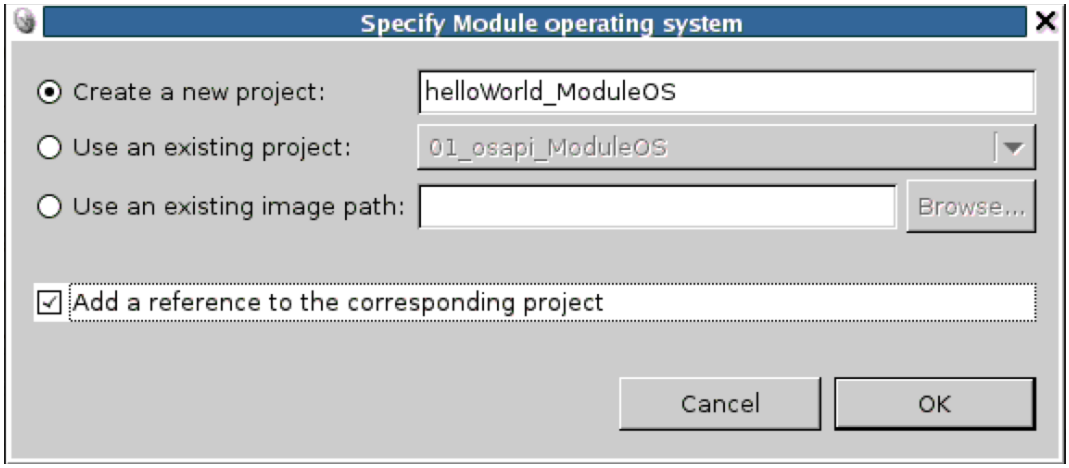


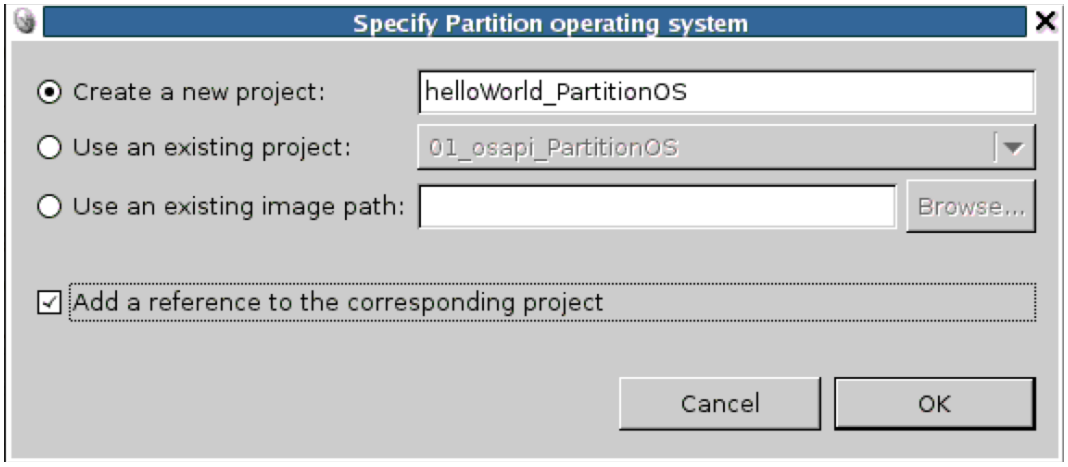

e. Select the appropriate Board Support package. Make sure the debug Build spec is selected. This example assumes the fsl_p2020_rdb_AMP_CORE0 board support package is selected.



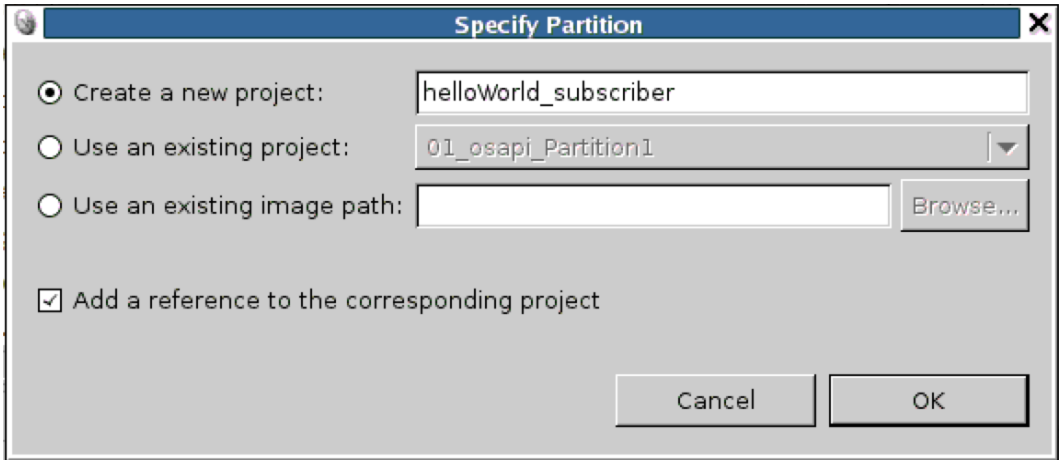
f. Select the default options for adding the ConfigRecord, ModuleOS, and PartitionOS. Make sure the "**Add a reference to the corresponding project**" check box is selected.



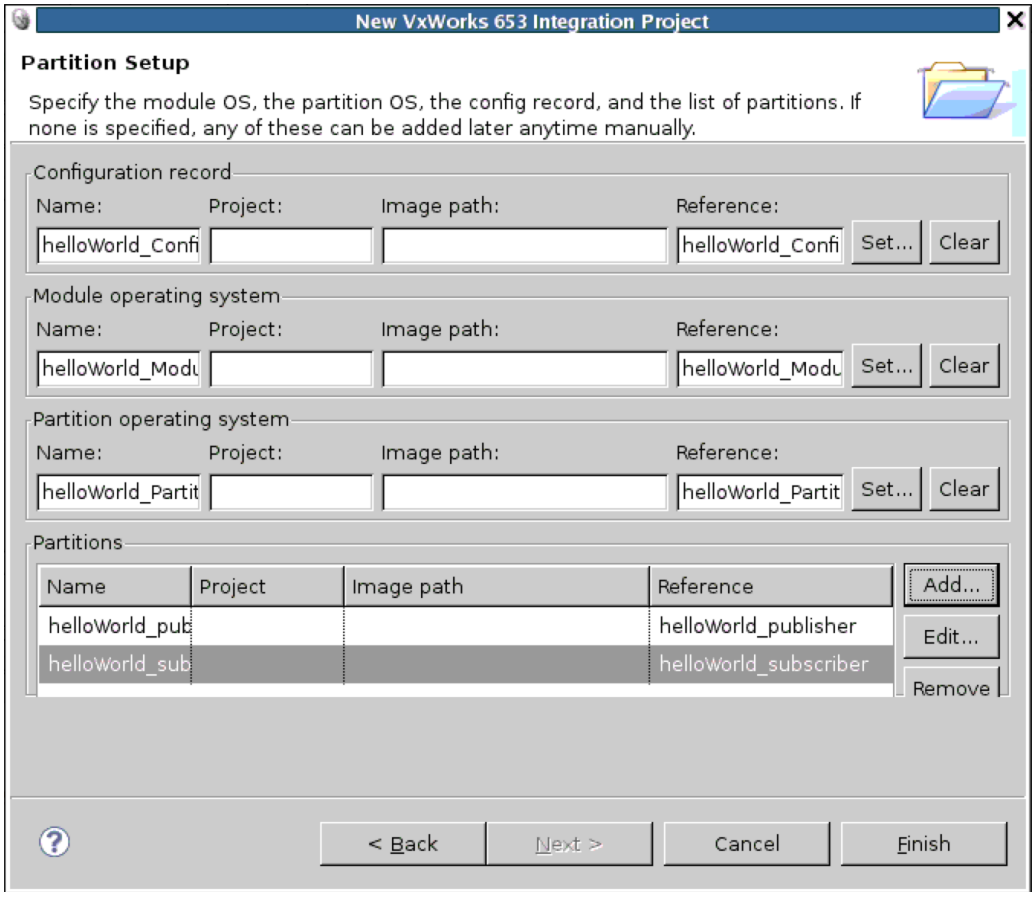



g. Create two partitions, **helloWorld_publisher** and **helloWorld_subscriber**, to create a Publisher and a Subscriber application, respectively. Make sure the **"Add a reference to the corresponding project"** check box is selected.

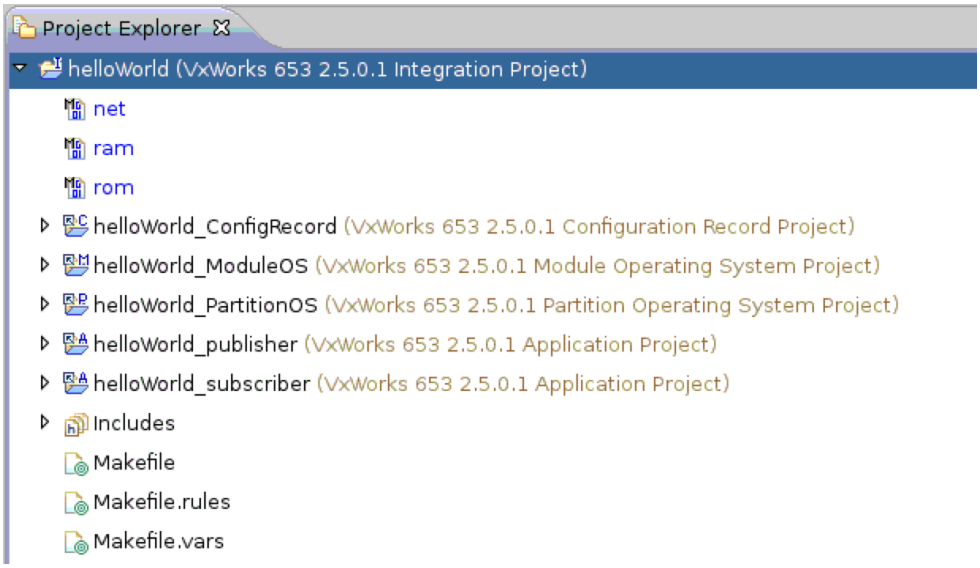



h. Now you are ready to create the Integration Project.

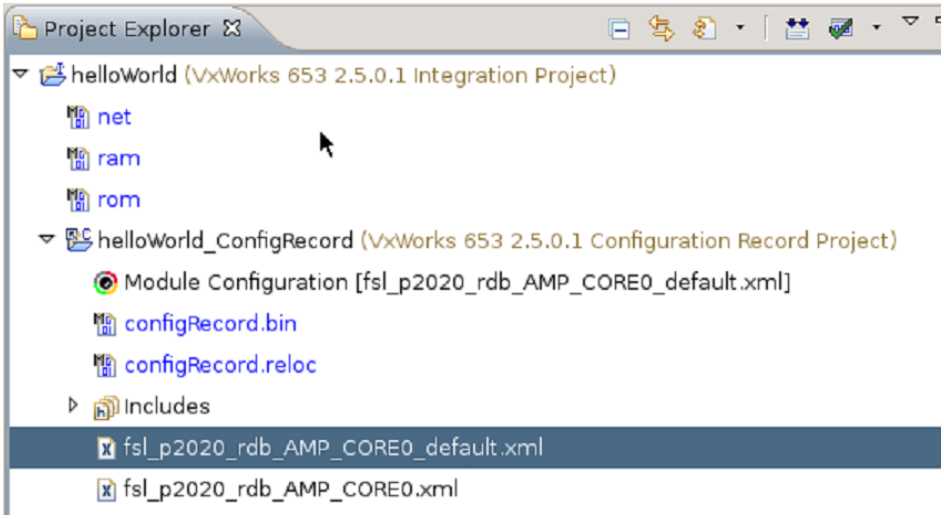



i. Click **Finish** to create the Integration project.

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.

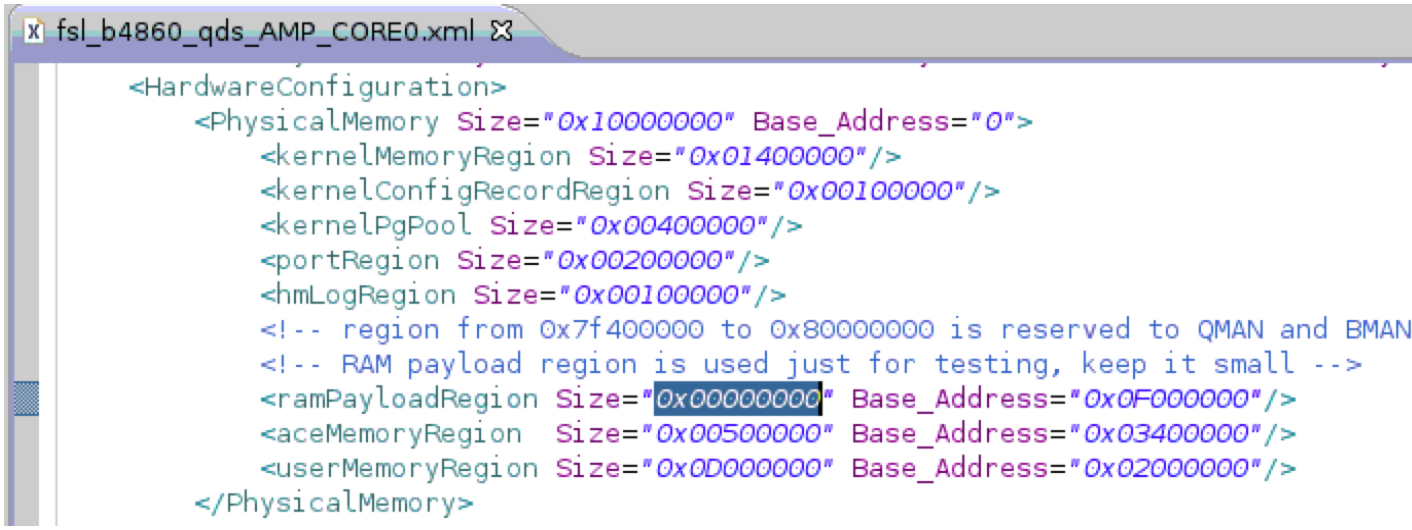
f. Under Payloads:

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

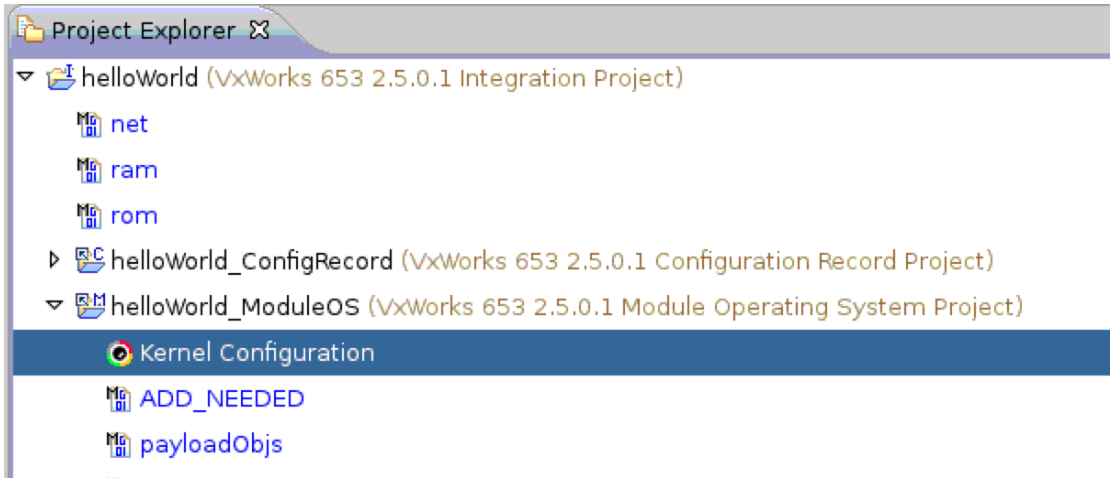
ii. Create a copy of the **PartitionPayload**, and change **NameRef** to **helloWorld_subscriber**.

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]**.

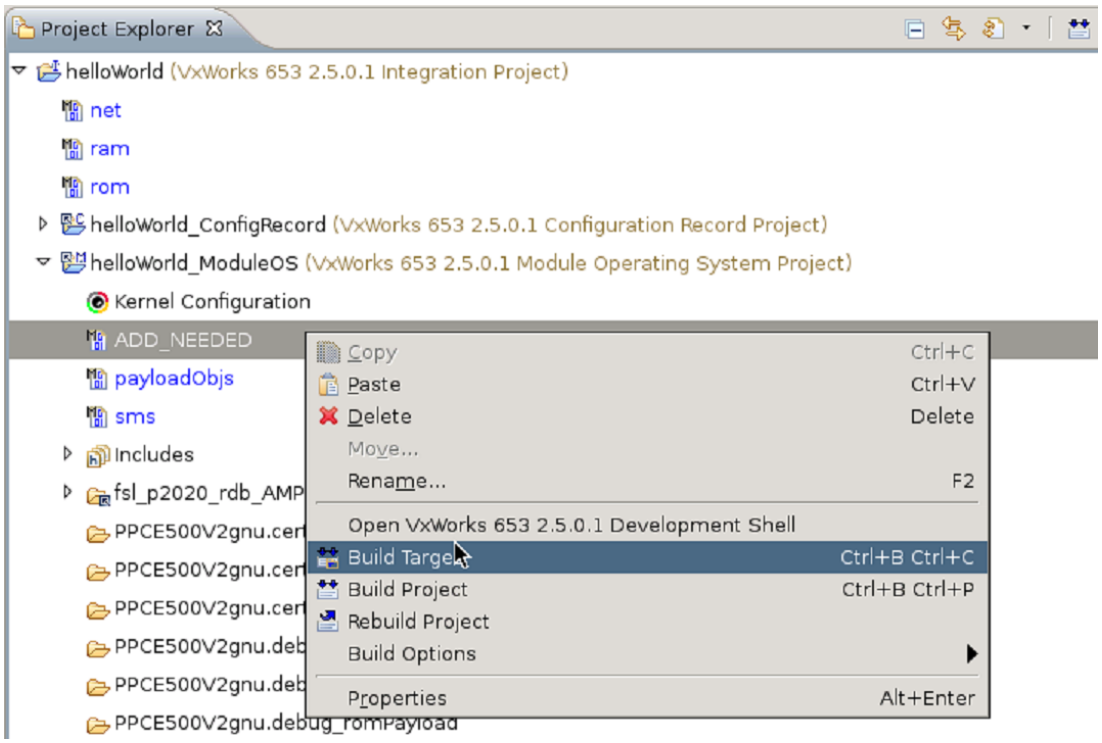
b. Include **development tool components->debug utilities [INCLUDE_DEBUG_UTIL]**. This is needed to enable worker tasks.

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 and Utilities Custom Support for VxWorks 652 Version 2.5.0.1 Platforms* (**RTI_CoreLibrariesAndUtilities_PlatformNotes_VxWorks653_v2.5.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++:

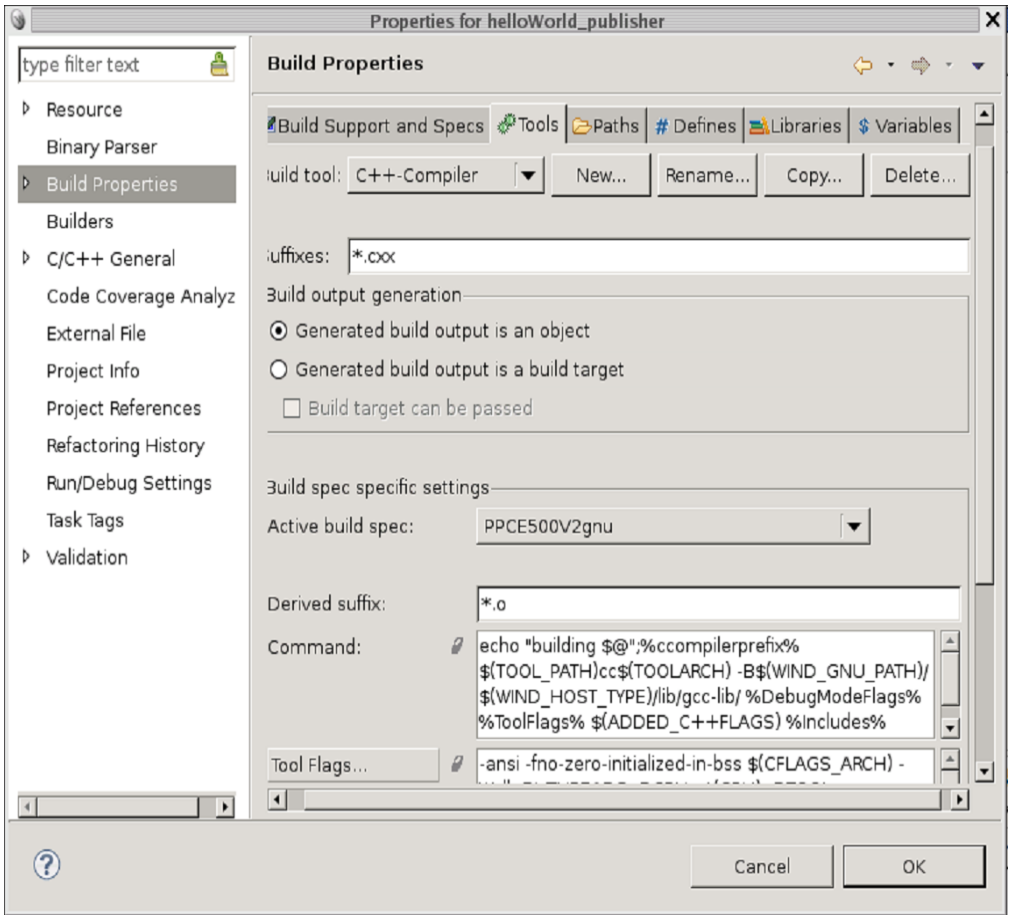
```
rtiddsgen -language C++ -example ppce500v2Vx653-2.5gcc4.3.3 HelloWorld.idl
```

For more information on the ppce500v2Vx653-2.5gcc4.3.3 architecture, please see the separate document, Custom Support for VxWorks 653 Version 2.5.0.1 Platforms.

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**.

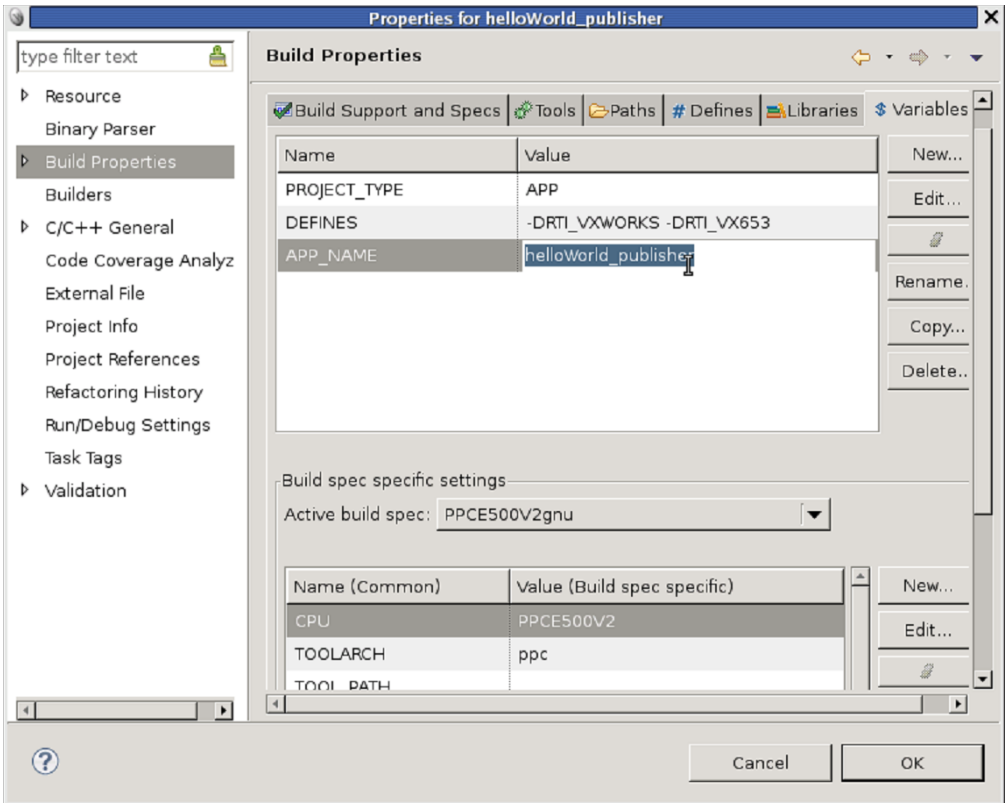
iv. Click **OK**.




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

c. Select **Properties**.

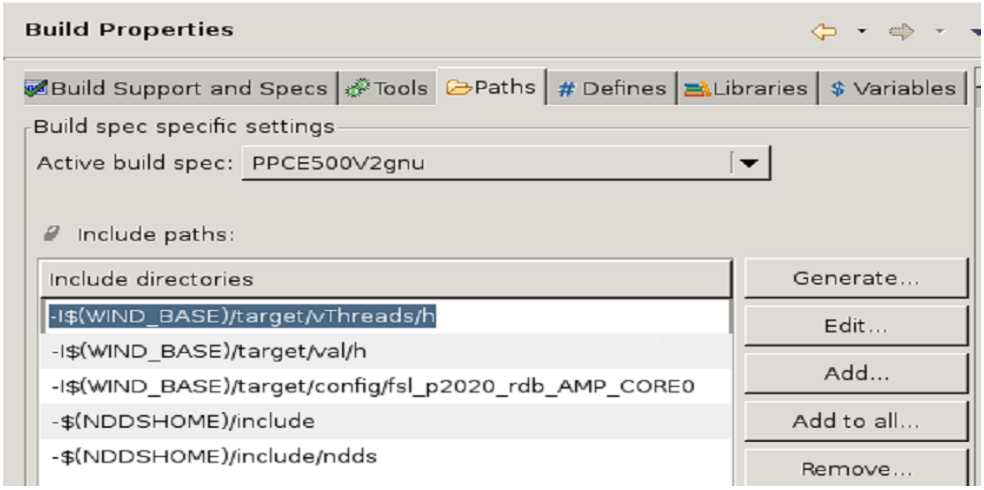
d. In the Variables tab, add -DRTI_VXWORKS -DRTI_VX653 to DEFINES.




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:

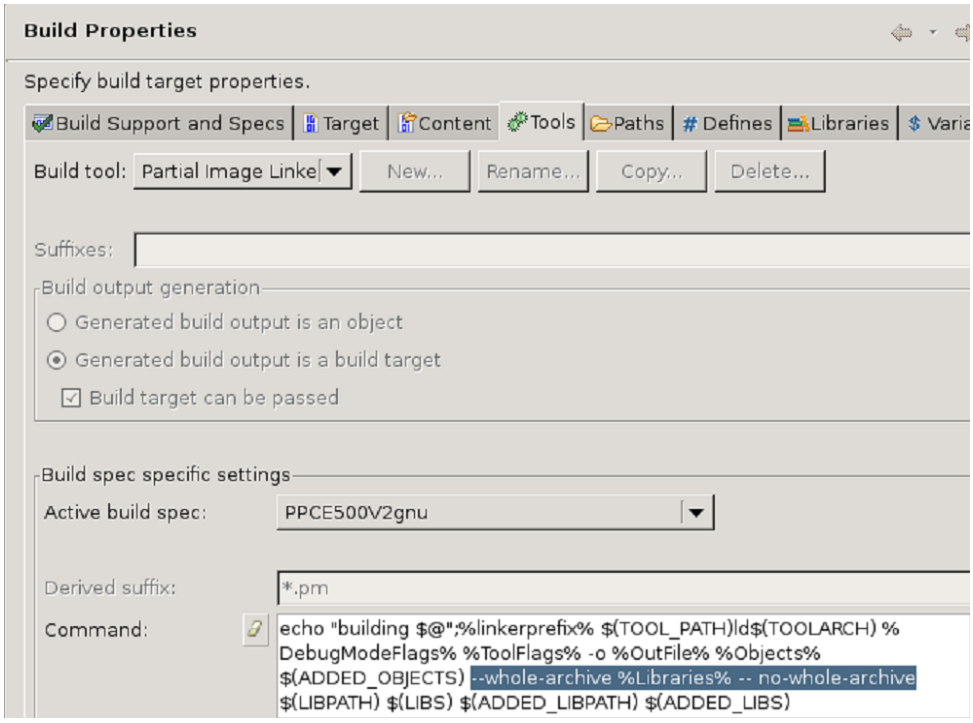
**For C++:**

```
$(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:**

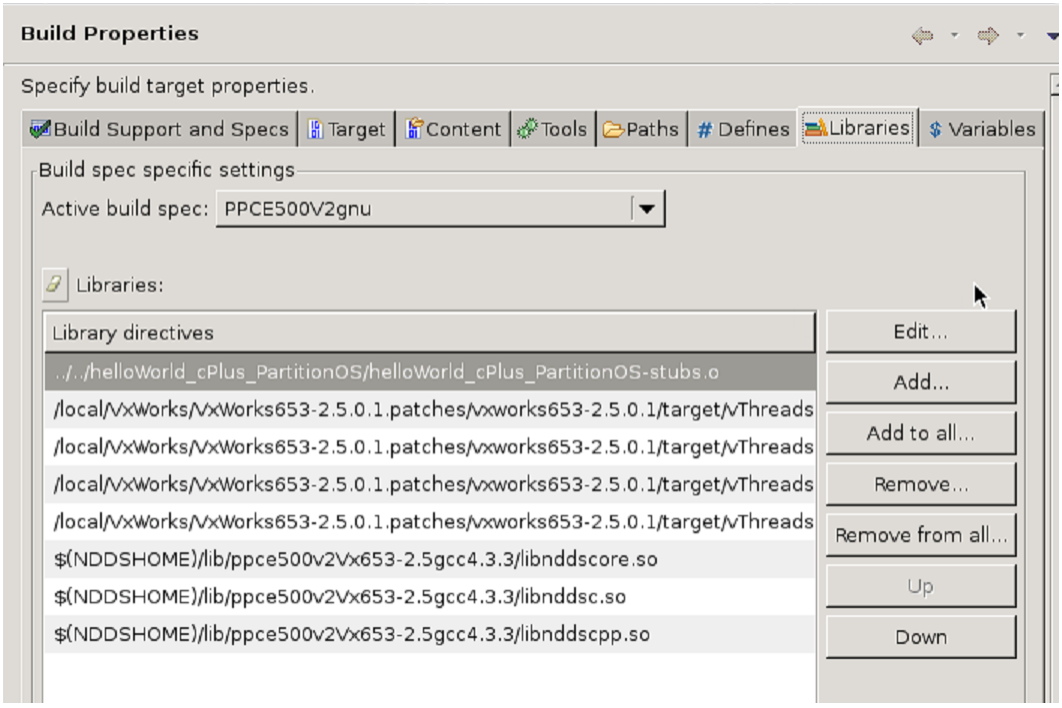
```
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddscore.so
$(NDDSHOME)/lib/ppce500v2Vx653-2.5gcc4.3.3/libnddsc.so
```

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:

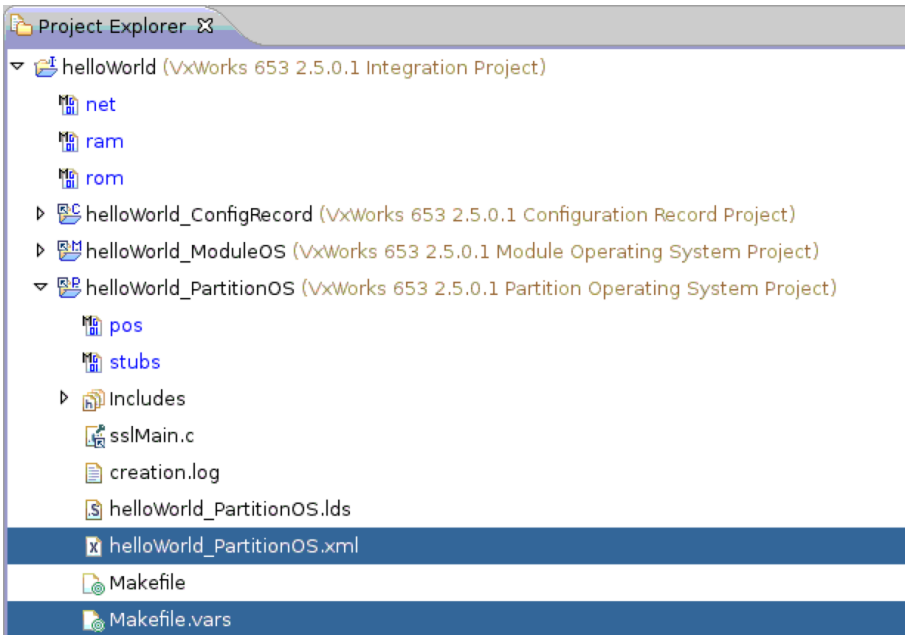



h. Repeat the same process for **helloWorld_subscriber**.

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 on a b4860 QDS 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 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...
```

# Chapter 8 Getting Started on Wind River VxWorks MILS 2.1.1 Systems

To use Connext DDS on a VxWorks MILS 2.1.1 system, you must have the patch provided by Wind River that corrects defect number WIND00343321. You can obtain this patch through the regular Wind River support channel.

This section provides instructions to configure a complete MILS 2.1.1 system image with an application that uses Connext DDS. Please refer to the documentation provided by Wind River for more information on the MILS system; you should also refer to the *RTI Connext DDS Core Libraries Platform Notes*.

This section will guide you through the process of generating, compiling, and running a "Hello, World" application on VxWorks MILS 2.1.1 systems 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.

The instructions in this chapter use Wind River Workbench to create the MILS system image. The overview of the workflow includes:

- Step 1: Generate Support Files and Example with rtiddsgen (Section 8.1 on the next page)
- Step 2: Create a VxWorks GuestOS Application Project (Section 8.2 on the next page)
- Step 3: Create a VxWorks MILS Integration Project (Section 8.3 on page 70)
- Step 4: Integrate GuestOS Application Project and Generated rtiddsgen Files into MILS Integration Project (Section 8.4 on page 75)
- Step 5: Deploy MILS Image to Target (Section 8.5 on page 76)

## 8.1 Step 1: Generate Support Files and Example with rtiddsgen

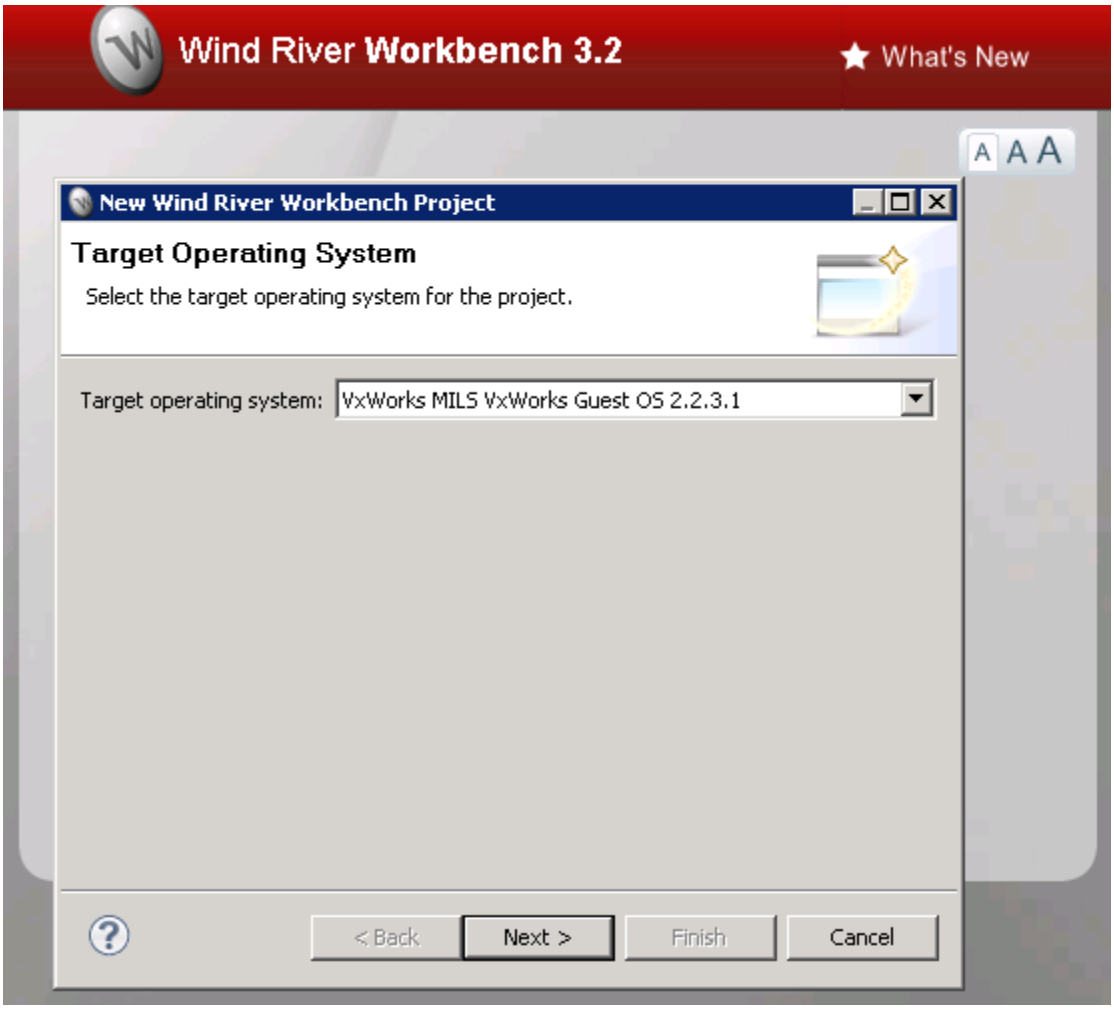
1. Given a sample file with an IDL definition, obtain the Connext DDS support files and example by following the steps in Building and Running Hello World, in the RTI Connext DDS Core Libraries Getting Started Guide".

   After you have completed this step, you will end up with source files and headers that implement the type support for your IDL definition, as well as an example publisher and an example subscriber for the type.

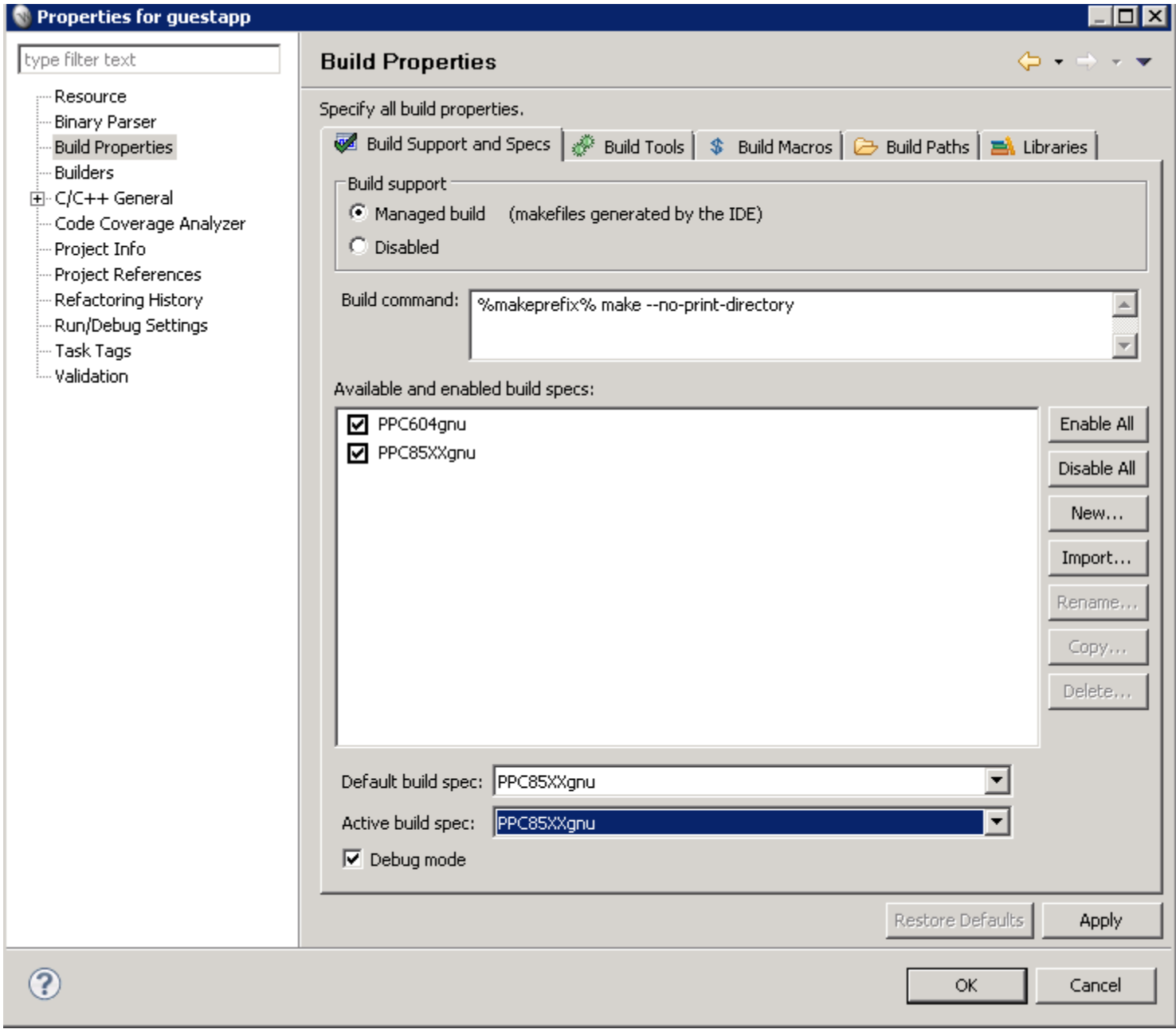
   Eventually, when we create our MILS application, we will call the **publisher_main** or **subscriber_main** functions from it, so make sure the **publisher_main** or **subscriber_main** functions are not declared as "static" (modify the example publisher and subscriber if you need to by simply removing the "static" qualifier from their function definitions if they have it).
2. If you are using C++, rename all **.cxx** files to **.cpp**.

## 8.2 Step 2: Create a VxWorks GuestOS Application Project

a.
   1. From the **File** menu, select **New, Wind River Workbench Project**.
   2. In the resulting dialog, select **VxWorks MILS VxWorks Guest OS 2.2.3.1** and click **Next**.




3. In the **Build Type** dialog, select **Application** and click **Next**.
4. For the project name, type **guestapp**, and click **Finish**.
5. Right-click on the project and select **Properties**.
6. In the left pane of the **Properties** dialog, select **Build Properties**.
7. In the **Build Support and Specs** tab, set the **Default build spec and** the **Active build spec** to **PPC85XXgnu**.




8. In the Build Macros tab:
    a. Under **Build Macro Definitions**, set **DEFINES** to **-DRTI_VXWORKS**.
    b. Define a new build macro named **NDDSHOME** (click **New** to add it). Set its value to the path where Connext DDS is installed. If you are using a Windows system, use

quotes around the path (for example, the value could be **"C:\Program Files\rti_connext_dds-5.3.0"**).

c. Set **Active build spec** to **PPC85XXgnu**.

d. In **CFLAGS_ARCH**, add **-mlongcall** to the end (leave the rest of the flags in that cell as is).

e. Set **LIBPATH** to **-L$(NDDSHOME)/lib/<architecture>**.

f. If you will *not* be using C++: set **LIBS** to: **-lnddscz -lnddscorez**.

If you will be using C++: set LIBS to **-lnddscppz -lnddscz -lnddscorez**

g. Set **NET_OBJS** to the following (all in one line):

```
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsNetwrsComponent.o
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsNetinetComponent.o
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsNetCommonComponent.o
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsNetBufCommonComponent.o
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsNetUtilComponent.o
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx/avlLib.o
```

h. If you will *not* be using C++: no changes are needed to **GOS_OBJS**.

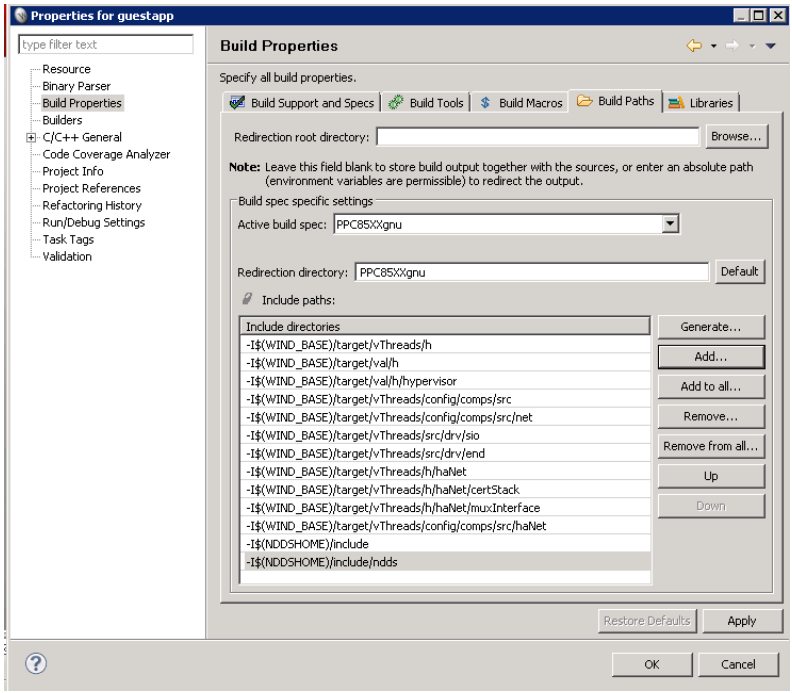
If you *will* be using C++: append the following to **GOS_OBJS** (all in one line):

```
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsCplusComponent.o
$(WIND_BASE)/target/vThreads/lib/obj$(CPU)
gnuvx/vThreadsCplusLibraryComponent.o
```

1. In the Build Paths tab:

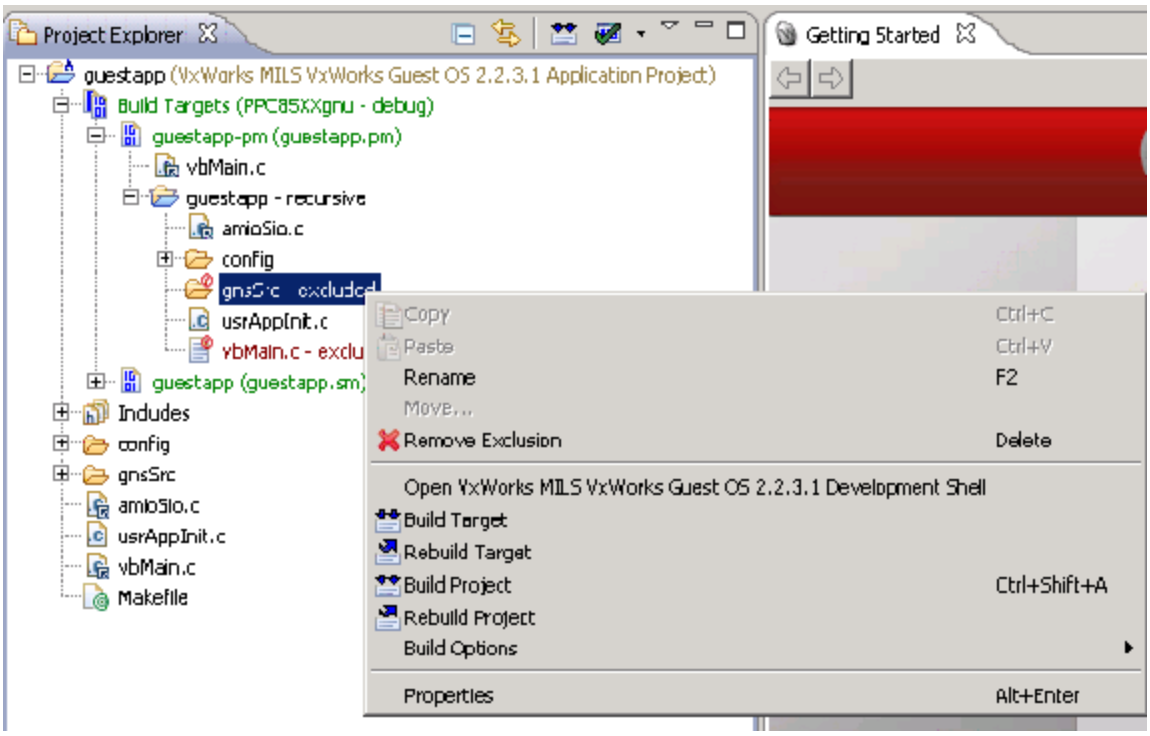
   a. Add **-I$(NDDSHOME)/include**

   b. Add **-I$(NDDSHOME)/include/ndds**.

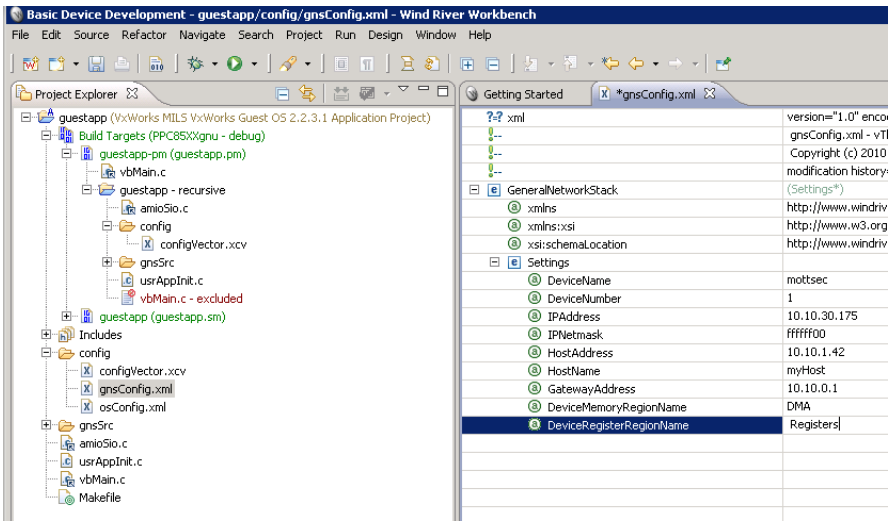



c. Click **OK**. If you see a prompt about rebuilding the C/C++ index, click **Yes**.

2. In the Project Explorer pane on the left, expand **Build Targets**, expand **<project_name>-pm**, and finally expand **<project_name>-recursive**. Then right-click on the **gnsSrc-excluded** item and remove the exclusion for it. Leave the rest in the default include/exclude mode.



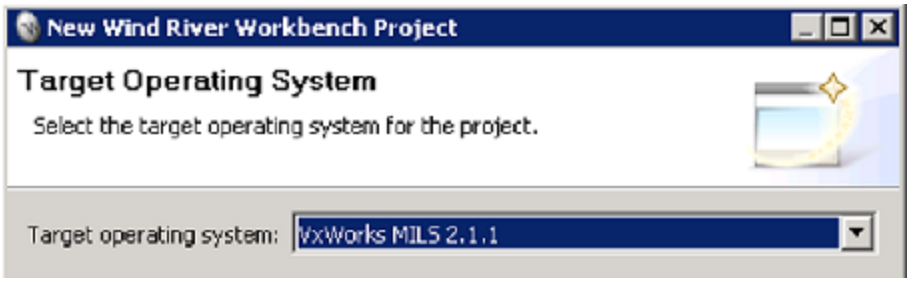

3. Configure the network for your setup (open the top-level **config** directory of your VxWorks GuestOS project and edit the **gnsConfig.xml file** to match your network setup).



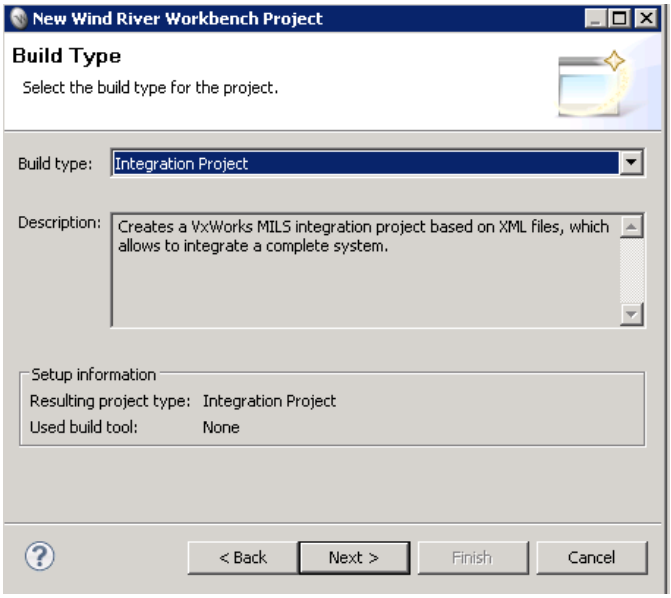

4. Right-click on the project and build it. If you see a dialog asking if you want to set the include search path, click **Continue**.

## 8.3 Step 3: Create a VxWorks MILS Integration Project

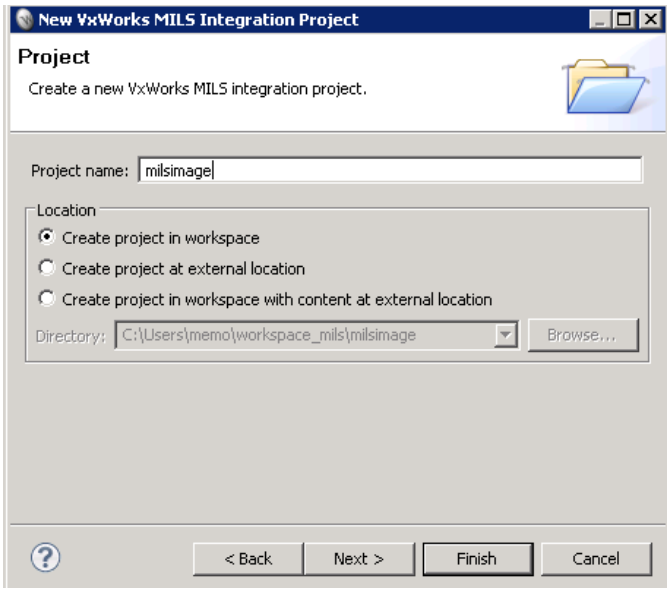
1. From the **File** menu, select **New, Wind River Workbench project**.
2. In the resulting dialog, select **VxWorks MILS 2.1.1**.



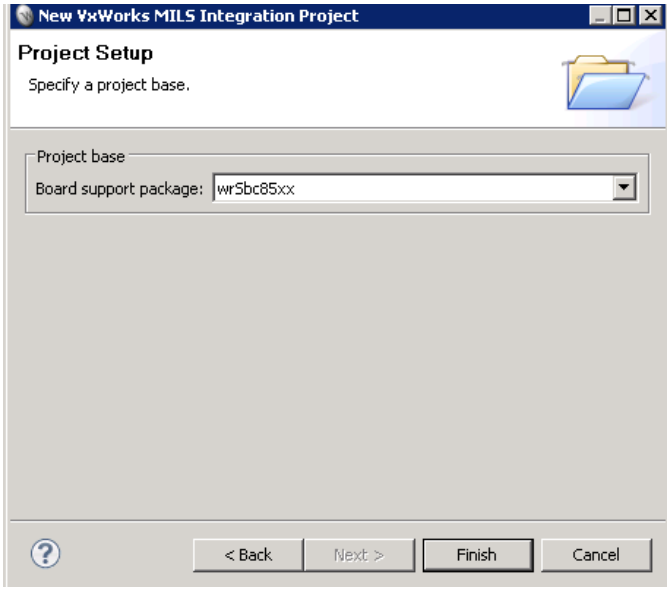

3. For **Build Type**, select **Integration Project** and click **Next**.



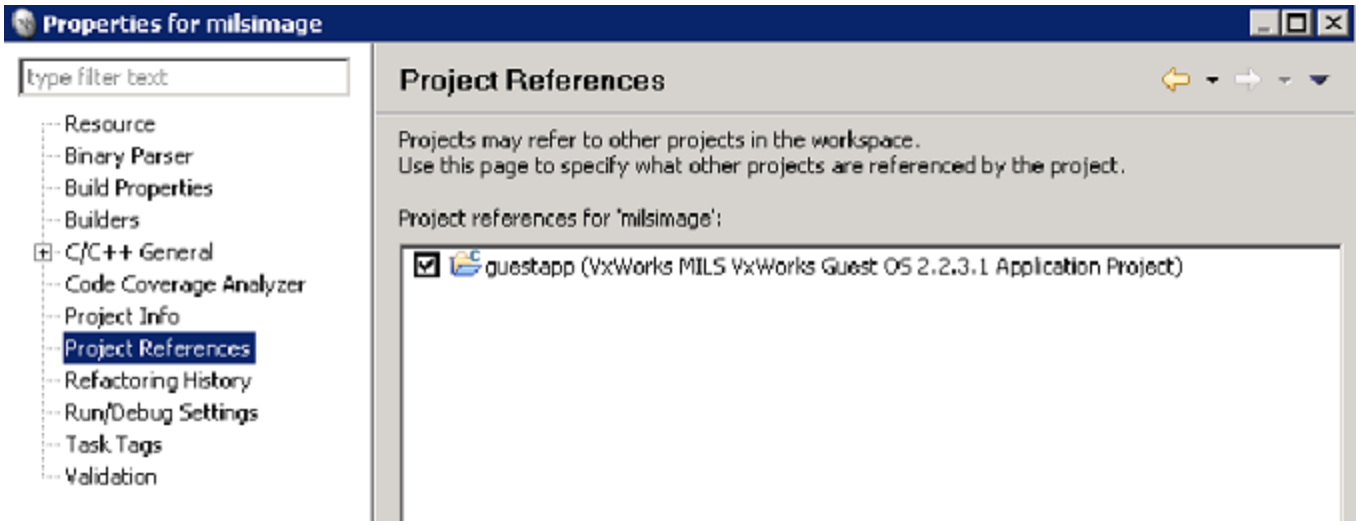
4. In the Project dialog: for **Project name**, type **milsimage** and click **Next**.



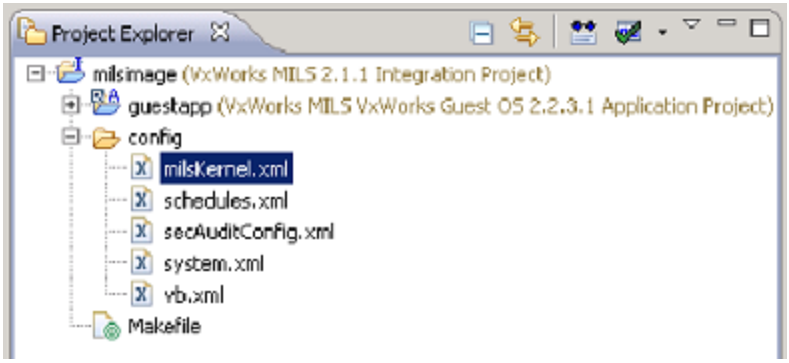

5. In the Project Setup dialog: for **Board Support Package**, select **wrSbc85xx** and click **Finish**.




6. Right-click on the newly created **milsimage** project and select **Properties**.
7. In the Project References section, add **guestapp** (the VxWorks GuestOS project that you created earlier). Once you accept these changes, you should see your **milsimage** and **guestapp** projects merge into a single entity in the Project Explorer.




8. In the Project Explorer pane on the left, navigate to the **config** directory of the **milsimage** project, where you will find the file, **milsKernel.xml**.




9. Double-click on **milsKernel.xml** and use the XML editor to make these changes:
   - Under MILS Kernel, set **RamSize** to **0x0A00000**.
   - Under RamPayload, set **RamPayloadSize** to **0x17000000**.
   - Under PcbMemPool, set **PcbPoolAddr** to **0xA00000**.
   - Under PayloadsMemPool, set **PayloadsMemPoolAddr** to **0xB00000**.
   - Under PayloadsMemPool, set **PayloadsMemPoolSize** to **0x8000000**.
   - Under SharedMemPool, set **SharedMemPoolAddr** to **0x8B00000**.
   - Under SharedMemPool, set **SharedMemPoolSize** to **0x4000**.

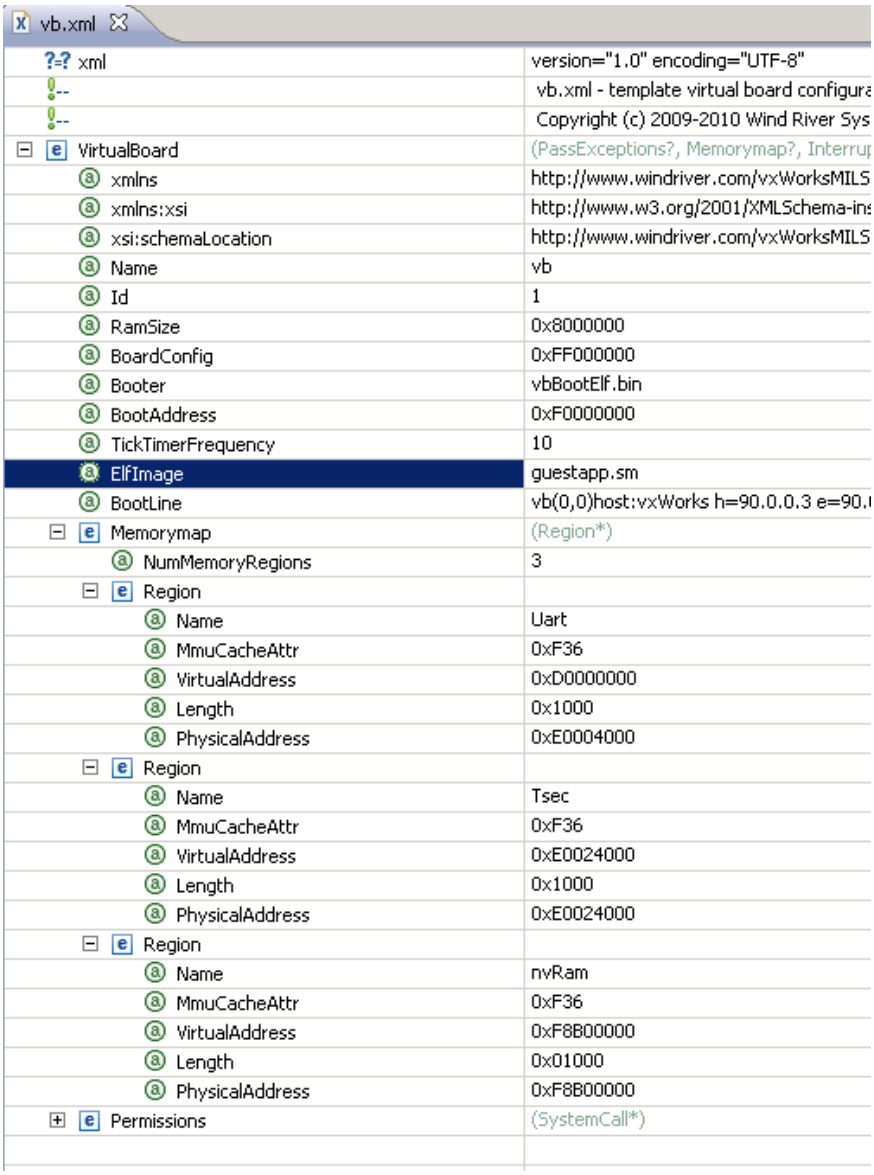
   **Note:** After changing the value of a cell, you must move to another cell so that your change will be picked up.
10. In the Project Explorer pane on the left, navigate to the **config** directory of the **milsimage** Integration project, where you will find the file, **vb.xml**.

11. Double-click on **vb.xml** and use the XML editor to make these changes:
    - Under VirtualBoard, set **RamSize** to **0x8000000**.
    - Under VirtualBoard, set **ElfImage** to **guestapp.sm**. Note that the file extension is **.sm**, not **.pm**. Also, if you used a different name for the GuestOS application project, you will need to modify this value accordingly.
    - Right-click on the **VirtualBoard** element and select **Add Child, Memory Map** (because we need to map in some devices).
12. In the newly created Memory-Map element:
    - Set **NumMemoryRegions** to **3**.
    - Right-click on the **Memory Map** element and select **Add Child, Region**; do this three times to add three regions.

      Make the following changes in each region:

| | Region 1 | Region 2 | Region 3 |
|---|---|---|---|
| **Name** | Uart | Tsec | nvRam |
| **MmuCacheAttr** | 0xF36 | 0xF36 | 0xF36 |
| **VirtualAddress** | 0xD0000000 | 0xE0024000 | 0xF8B00000 |
| **Length** | 0x1000 | 0x1000 | 0x01000 |
| **PhysicalAddress** | 0xE0004000 | 0xE0024000 | 0xF8B00000 |

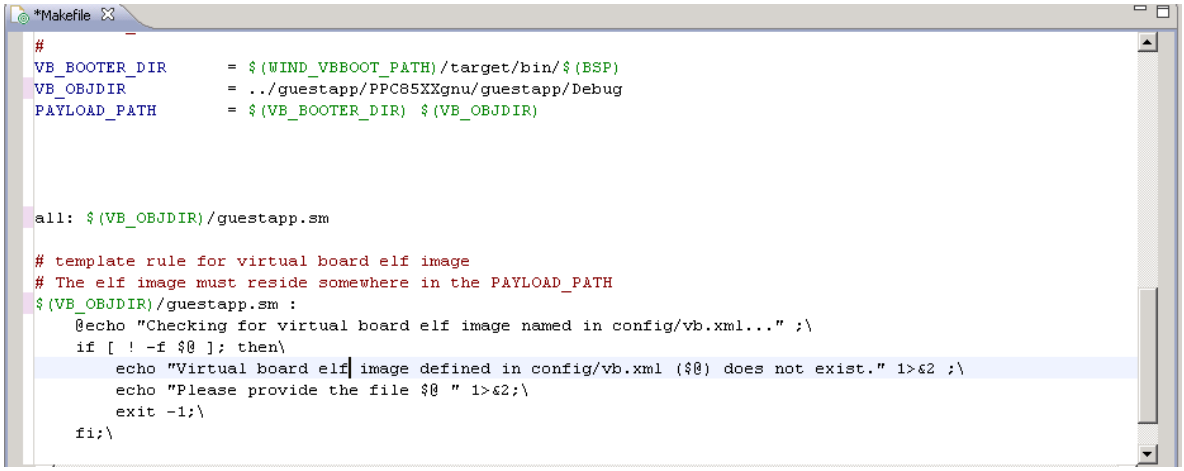
13. Save your changes (be sure to switch to another cell after you edit each cell's contents and before closing the file, so it registers all your changes).




14. Open the **integration** project's **Makefile** (it should be under the top-level **milsimage** project; do not confuse it with the **guestapp** project's **Makefile**).
15. Update the **VB_OBJDIR** make variable to:

    **../guestapp/PPC85XXgnu/guestapp/Debug**

    This way we point to the output of the **guestapp** application project.
16. Update the **all** make target to **$(VB_OBJDIR)/guestapp.sm**.
17. Update the rule after the **all** target so it also references **guestapp.sm** instead of **hello.sm**.




## 8.4 Step 4: Integrate GuestOS Application Project and Generated rtiddsgen Files into MILS Integration Project

1. Import the source files that you generated from your IDL file into the project:
   a. Right-click on the **guestapp** project and select **Import...**
   b. In the dialog, select **General, File System**. Navigate to the directory that contains your generated files.
   c. Click on the directory's name in the left pane of the resulting dialog box and check all the C/C++ source files and header files from that directory.
   d. Click **Finish**.
2. If you are using C++, rename all imported **.cxx** files to **.cpp** if you haven't already done so.
3. Make sure you have removed the **static** qualifier from the signature of the functions **publisher_main** and **subscriber_main** if they had it. These functions would be in the imported **<***idl_struct_name***>publisher.c** and **<***idl_struct_name***>subscriber.c** files, respectively. The objective is to make them callable from outside these files.
4. Using the Project Explorer in the left pane of WorkBench, navigate to the file **usrAppInit.c** in the **guestapp** project. Double-click to edit the file and replace its entire contents with the following:

```
#include <stdio.h>
#include <taskVarLib.h>
#include <muxLib.h>
#include <bootLib.h>
#include <routeLib.h>
#include <netShow.h>
#include <usrLib.h>

/* defines */
```

```
#undef DEBUG


/* globals */
UINT32 boardNum = 0;
extern BOOT_PARAMS sysBootParams;
extern void usrNetworkInit (void);
extern int publisher_main(int domainId, int sample_count);


void usrAppInit (void)
{
       char dev[END_NAME_MAX + 2]; /* device name + unit */
         taskVarInit( );
         boardNum = vbConfig->boardID;
         /* avoid startup messages in console */
         taskDelay (sysClkRateGet () * 10);
         printf ("\n\n*** MILS User Space RTI App.***\n\n");
         printf ("On Virtual Board %d\n\n",boardNum);
         /* start the network */
         usrNetworkInit ();
         routeAdd ("0.0.0.0", sysBootParams.ead);
         taskDelay (sysClkRateGet () * 1);
         printf("\n\n   RTI App Starting   \n\n");
         sprintf(dev,"%s%d\n", sysBootParams.bootDev,
                     sysBootParams.unitNum);
         printf("before ifShow\n");
         ifShow(dev);
         printf("after ifShow\n");
         muxShow (sysBootParams.bootDev,sysBootParams.unitNum);
         printf ("\nAPPLICATION: Launching\n\n");
         taskSpawn ("pub", 75, 0, 0x20000, (FUNCPTR)publisher_main,
                        0, 100, 0, 0, 0, 0, 0, 0, 0, 0);
         while (1){
             taskDelay (sysClkRateGet () * 60);
             printf ("\nVirtual Board %d is alive.\n", boardNum);
             /*memShow(0);*/
         }
}
```

5. Build the MILS Integration project: right-click the **milsimage** project and select **Rebuild** from the context menu. The build should complete with no errors.

## 8.5 Step 5: Deploy MILS Image to Target

Once you have completed Step 4, you should have a MILS image file in your file system. The location where you can find the image file relative to the MILS integration project is: **obj_wrSb-c85xx/milsKernel.elf**.

Upload this **.elf** image file to your target. One way to do this is to upload the file to a tftpserver that is accessible from your target board, then have the target board pull the image over the network. Boards with a VxWorks boot loader can do this in a standard way; consult the board's documentation for further information.

Once you have deployed the MILS image to your board, it will start publishing samples with your IDL definition as a data type. It will print out to the target's console as it publishes samples. You can start a subscriber in the same domain on other computers connected to the same network to verify the samples are being sent.